FreeBSD™ JOURNAL
Nov/Dec 2017

Monitoring & Metrics

BORN TO DISRUPT

MODERN. UNIFIED. ENTERPRISE-READY.

INTRODUCING THE TRUENAS® X10, THE MOST COST-EFFECTIVE ENTERPRISE STORAGE ARRAY ON THE MARKET.

Perfectly suited for core-edge configurations and enterprise workloads such as backups, replication, and file sharing.

- ★ **Modern:** Not based on 5-10 year old technology (yes that means you legacy storage vendors)
- ★ **Unified:** Simultaneous SAN/NAS protocols that support multiple block and file workloads
- ★ **Dense:** Up to 120 TB in 2U and 360 TB in 6U
- ★ **Safe:** High Availability option ensures business continuity and avoids downtime
- ★ **Reliable:** Uses OpenZFS to keep data safe
- ★ **Trusted:** Based on FreeNAS, the world's #1 Open Source SDS
- ★ **Enterprise:** 20TB of enterprise-class storage including unlimited instant snapshots and advanced storage optimization for under $10,000

The new TrueNAS X10 marks the birth of a new entry class of enterprise storage. Get the full details at iXsystems.com/TrueNAS.

iXsystems™

Table of Contents

Nov/Dec 2017

Monitoring & Metrics

COLUMNS & DEPARTMENTS

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

S&W Publishing LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board

Happy Holidays

Wishing You a Joyful New Year •

2018

George Neville-Neil
President of the *FreeBSD Foundation* Board of Directors

SEE TEXT ONLY

Monitoring ZFS

BY ALLAN JUDE

ZFS is an advanced file-system, but it is also one of the most observable. The combination of static and dynamic DTrace probes, statistics, and tooling built into ZFS means it is one of the easiest filesystems to do performance analysis on. This article covers checking the health of the pool, measuring load and performance in real time, and using historical statistics to detect changes in health, performance, or behavior. Then provides a brief overview of other tools to consider for monitoring a system in even more detail. First up, is the pool healthy?

HEALTH CHECKING • zpool Status

The zpool status command is the first place to look to assess the general health of the pool. It prints a visual representation of the layout of the devices in the pool, and the status of each device. In addition to the status of each device, there are also columns of counters, showing the number of read, write, and checksum errors that have been detected on each device.

Each device can be in one of these states:

- **Online** — The device is healthy and working as expected.
- **Offline** — An administrator has marked this device as offline.
- **Degraded** — The device is functioning at a reduced capacity. Usually only applies to a top level vdev, like RAID-Z or a Mirror; indicates that one or more member disks has failed and the system is using parity to remain operational.
- **Faulted** — The device or pool is no longer working because too much data is missing. If a device or pool loses more devices than it has redundancy, files may be inaccessible or lost. Try to reconnect the missing devices to continue.
- **Removed** — An underlying device has been removed. This can happen when a disk fails and the device is removed by the operating system, or when a disk is physically disconnected by an operator.
- **Missing** — ZFS was unable to find, or unable to open, the device. Try to reconnect the device, or solve the error that prevented ZFS from opening the device (such as it being in use by another process). The zpool online command is useful to bring a device back online.
- **Replacing** — A device is being replaced. When replacing a device that is online, a new top-level vdev called replacing-X (where X is an incrementing integer) will be created; it is effectively a mirror with the new and old devices as members. Data is copied from the old device to the new device. Once the operation is complete, the old device will

be detached from the pool, and the new device will become a regular member of the vdev.
• Spare — The device is missing or otherwise degraded and has been temporarily replaced with a spare.
• Resilvering — This device was temporarily offline or has suffered some corruption and the missing or damaged data is being replaced from available parity.

```
# zpool status
  pool: media
 state: ONLINE
  scan: resilvered 25.7M in 0h0m with 0 errors on Sat Oct 14 14:40:18 2017
config:

        NAME                      STATE     READ WRITE CKSUM
        media                     ONLINE       0     0     0
          raidz2-0                ONLINE       0     0     0
            gpt/s5-Z5009MV3       ONLINE       0     0     0
            gpt/s3-Z500Z78C       ONLINE       0     0     0
            gpt/s2-Z500ZKL8       ONLINE       0     0     0
            gpt/s4-Z503E2PR       ONLINE       0     0     0
            gpt/s1-Z1F3134B       ONLINE       0     0     0
            gpt/s6-Z500XXPA       ONLINE       0     0     0

errors: No known data errors
```

If one or more problems are detected with the pool, a summary will be displayed at the end of the status output. Running zpool status -v <optional poolname> will extend this summary, and provide a list of each file that has suffered damage, allowing those individual files to be restored from backups.

```
  # zpool status -v zroot
    pool: zroot
   state: DEGRADED
  status: One or more devices has experienced an error resulting in data
          corruption.  Applications may be affected.
  action: Restore the file in question if possible.  Otherwise restore the
          entire pool from backup.
     see: http://illumos.org/msg/ZFS-8000-8A
    scan: resilvered 1.43T in 79h21m with 3 errors on Sat Oct 14 01:18:56 2017
  config:
         NAME                        STATE     READ WRITE CKSUM
         zroot                       DEGRADED   113     0     0
           raidz1-0                  DEGRADED   113     0     0
             ada0p3                  ONLINE       0     0     0
             ada1p3                  ONLINE       0     0     0
             ada2p3                  ONLINE     113     0     0
             replacing-3             OFFLINE      0     0     0
               17161359962879308376  OFFLINE      0     0     0
               ada3p3                ONLINE       0     0     0


  errors: Permanent errors have been detected in the following files:

          /usr/src/contrib/binutils/ld/emultempl/armcoff.em
          /usr/src/contrib/binutils/ld/emultempl/armelf.em
          /usr/src/contrib/binutils/ld/emultempl/astring.sed
          /usr/src/contrib/binutils/opcodes/ChangeLog-2006
```

The zpool status command also tracks the progress of scrub and resilver operations, including an average speed and a completion estimate. The speed estimate is an average for the entire operation, which is much slower for the first few %, so consider waiting until 5%–10% completion before taking the speed and ETA seriously. If the system is rebooted or otherwise interrupted during the resilver operation, the estimate may be stuck at 1 byte per second for a long time.

```
# zpool status -v zroot
  pool: zroot
 state: ONLINE
  scan: scrub in progress since Wed Oct 18 22:27:19 2017
        20.7G scanned out of 32.1G at 401M/s, 0h0m to go
        0 repaired, 64.50% done
config:

        NAME                      STATE     READ WRITE CKSUM
        zroot                     ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            gpt/i1-14450DAFF1A8   ONLINE       0     0     0
            gpt/i2-154310EB96A5   ONLINE       0     0     0
```

S.M.A.R.T.

Disks also provide some insight into the state of their health using the SMART (Self-Monitoring, Analysis and Reporting Technology) protocol. For spinning disks, the greatest indicators of impending failure are usually the number of pending and reallocated sectors. Each manufacturer provides a different set of statistics, so it is difficult to create hard and fast rules about what means the disk is underperforming or indi-

cating potential failure. To make the most sense out of the various counters in the SMART status, you need a reference point, what those counters looked like in the past, how much and how fast they have changed. Another condition to watch out for is disks that have a high and rapidly growing cycle count. If the disk is going to sleep and waking up every few seconds, this will put tremendous wear on the disk. The disk may need to be given specific commands to adjust the idle timeout, or need updated firmware to fix the problem. SAS disks generally provide fewer counters but are more consistent across drive models and manufacturers.

```
# smartctl -a /dev/ada1
=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.14 (AF)
Device Model:     ST2000DM001-1CH164
Serial Number:    Z1E1DWN4
LU WWN Device Id: 5 000c50 04e53cf8d
Firmware Version: CC43
User Capacity:    2,000,398,934,016 bytes [2.00 TB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    7200 rpm
Form Factor:      3.5 inches
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ATA8-ACS T13/1699-D revision 4
SATA Version is:  SATA 3.0, 6.0 Gb/s (current: 3.0 Gb/s)
Local Time is:    Thu Nov 30 00:56:27 2017 UTC
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   113   099   006    Pre-fail  Always       -       53347512
  3 Spin_Up_Time            0x0003   095   095   000    Pre-fail  Always       -       0
  4 Start_Stop_Count        0x0032   100   100   020    Old_age   Always       -       9
  5 Reallocated_Sector_Ct   0x0033   100   100   036    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x000f   083   060   030    Pre-fail  Always       -       4510892460
  9 Power_On_Hours          0x0032   087   087   000    Old_age   Always       -       12060
 10 Spin_Retry_Count        0x0013   100   100   097    Pre-fail  Always       -       0
 12 Power_Cycle_Count       0x0032   100   100   020    Old_age   Always       -       9
183 Runtime_Bad_Block       0x0032   100   100   000    Old_age   Always       -       0
184 End-to-End_Error        0x0032   100   100   099    Old_age   Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
188 Command_Timeout         0x0032   100   100   000    Old_age   Always       -       0 0 0
189 High_Fly_Writes         0x003a   099   099   000    Old_age   Always       -       1
190 Airflow_Temperature_Cel 0x0022   074   065   045    Old_age   Always       -       26 (Min/Max 23/35)
191 G-Sense_Error_Rate      0x0032   100   100   000    Old_age   Always       -       0
192 Power-Off_Retract_Count 0x0032   100   100   000    Old_age   Always       -       9
193 Load_Cycle_Count        0x0032   100   100   000    Old_age   Always       -       269
194 Temperature_Celsius     0x0022   026   040   000    Old_age   Always       -       26 (0 19 0 0 0)
197 Current_Pending_Sector  0x0012   100   100   000    Old_age   Always       -       24
198 Offline_Uncorrectable   0x0010   100   100   000    Old_age   Offline      -       24
199 UDMA_CRC_Error_Count    0x003e   200   200   000    Old_age   Always       -       0
240 Head_Flying_Hours       0x0000   100   253   000    Old_age   Offline      -       12059h+46m+50.704s
241 Total_LBAs_Written      0x0000   100   253   000    Old_age   Offline      -       67220217075
242 Total_LBAs_Read         0x0000   100   253   000    Old_age   Offline      -       7368543577

SMART Error Log Version: 1
No Errors Logged
```

SATA SSDs usually have rather different SMART values since many of the regular values do not apply. Most SSDs will provide pairs of counters for the total amount of reads and writes that have been completed, allowing the administrator to track the wear lifetime of the device. Some SSDs even provide a drive lifetime statistic, as a %, either counting up or down toward the end of the useful life of the device. Sometimes the "raw value" has little meaning, and you need to look at the 'value' and 'thresh(old)' volumes instead. This SSD has relatively little wear:

```
#smartctl -a /dev/ada1
=== START OF INFORMATION SECTION ===
Model Family:     Intel 730 and DC S35x0/3610/3700 Series SSDs
Device Model:     INTEL SSDSC2BA200G4
Serial Number:    BTHV515103FW200MGN
LU WWN Device Id: 5 5cd2e4 04b7d7610
Firmware Version: G2010110
User Capacity:    200,049,647,616 bytes [200 GB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    Solid State Device
Form Factor:      2.5 inches
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ACS-2 T13/2015-D revision 3
SATA Version is:  SATA 2.6, 6.0 Gb/s (current: 6.0 Gb/s)
Local Time is:    Thu Nov 30 00:58:57 2017 UTC
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

SMART Attributes Data Structure revision number: 1
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0032   100   100   000    Old_age   Always       -       0
  9 Power_On_Hours          0x0032   100   100   000    Old_age   Always       -       18834
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       35
170 Available_Reservd_Space 0x0033   100   100   010    Pre-fail  Always       -       0
171 Program_Fail_Count      0x0032   100   100   000    Old_age   Always       -       0
172 Erase_Fail_Count        0x0032   100   100   000    Old_age   Always       -       0
174 Unsafe_Shutdown_Count   0x0032   100   100   000    Old_age   Always       -       20
175 Power_Loss_Cap_Test     0x0033   100   100   010    Pre-fail  Always       -       5290 (71 4859)
183 SATA_Downshift_Count    0x0032   100   100   000    Old_age   Always       -       0
184 End-to-End_Error        0x0033   100   100   090    Pre-fail  Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
190 Temperature_Case        0x0022   064   060   000    Old_age   Always       -       36 (Min/Max 31/40)
192 Unsafe_Shutdown_Count   0x0032   100   100   000    Old_age   Always       -       20
194 Temperature_Internal    0x0022   100   100   000    Old_age   Always       -       36
197 Current_Pending_Sector  0x0012   100   100   000    Old_age   Always       -       0
199 CRC_Error_Count         0x003e   100   100   000    Old_age   Always       -       0
225 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       655468
226 Workld_Media_Wear_Indic 0x0032   100   100   000    Old_age   Always       -       552
227 Workld_Host_Reads_Perc  0x0032   100   100   000    Old_age   Always       -       29
228 Workload_Minutes        0x0032   100   100   000    Old_age   Always       -       1129889
232 Available_Reservd_Space 0x0033   100   100   010    Pre-fail  Always       -       0
233 Media_Wearout_Indicator 0x0032   100   100   000    Old_age   Always       -       0
234 Thermal_Throttle        0x0032   100   100   000    Old_age   Always       -       0/0
241 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       655468
242 Host_Reads_32MiB        0x0032   100   100   000    Old_age   Always       -       275081

SMART Error Log Version: 1
No Errors Logged
```

INTERACTIVE PERFORMANCE MONITORING

Now that it is established that the pool is healthy, it is time to look at what is happening with the pool. These interactive monitoring tools shed light on the operations that are being performed in real time.

zpool iostat

The zpool iostat command will print data about the activity on the pool. It shows the number of read and write IOPS, as well as bytes per second. If no other parameters are given, it will display one status line for each pool. If a pool name is given, it will show only that pool. If an integer is given, it will run continuously, and print new stats every X seconds, where X is that integer. You'll notice the natural cycle of ZFS, where there are a minimal number of synchronous writes as requested by applications; then every 5 seconds all other buffered asynchronous writes are flushed out to disk. If you change the integer to a longer interval, it will provide an average over that time span.

```
# zpool iostat sestore5 1
              capacity     operations    bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
sestore5    46.0T  84.5T     99    189  14.8M  4.89M
sestore5    46.0T  84.5T      9    208  9.81M  1.09M
sestore5    46.0T  84.5T      0      0  31.9K      0
sestore5    46.0T  84.5T    103      0  12.9M      0
sestore5    46.0T  84.5T     64      0  7.79M      0
sestore5    46.0T  84.5T     38    570  4.74M  16.4M
sestore5    46.0T  84.5T     34    152  5.23M   826K
sestore5    46.0T  84.5T     11      0   278K      0
sestore5    46.0T  84.5T      6      0   247K      0
sestore5    46.0T  84.5T    146      0  16.4M      0
sestore5    46.0T  84.5T     31    977  1.46M  8.85M
sestore5    46.0T  84.5T      3      0   487K  3.99K
sestore5    46.0T  84.5T     35      0  4.12M      0
sestore5    46.0T  84.5T      1      0  2.00M      0
sestore5    46.0T  84.5T     12      0  63.9K      0
sestore5    46.0T  84.5T    244    650   978K  8.25M
sestore5    46.0T  84.5T      5      0   235K      0
sestore5    46.0T  84.5T      0      0      0      0


# zpool iostat sestore5 30
              capacity     operations    bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
sestore5    46.0T  84.5T     95    179  14.0M  4.93M
sestore5    46.0T  84.5T     22    163  5.43M  4.76M
sestore5    46.0T  84.5T      3    161  2.04M  5.96M
sestore5    46.0T  84.5T     17    264  3.71M  3.96M
sestore5    46.0T  84.5T     13    279  3.54M  4.01M
sestore5    46.0T  84.5T     21    416  4.87M  4.80M
sestore5    46.0T  84.5T     19    152  4.62M  6.89M
sestore5    46.0T  84.5T     36    166  6.21M  5.26M
sestore5    46.0T  84.5T     15    125  4.19M  2.66M
sestore5    46.0T  84.5T     16    136  3.54M  4.54M
```

top -m io

One of the fastest ways to figure out which application is causing all of the I/O is to use top. On FreeBSD top has a -m flag to change the mode. In IO mode, instead of tracking applications by CPU and memory usage, it tracks reads, writes, and other IO operations. This can help you determine which application is consuming all of the IO resources. To break things down further, see the section on the DTrace Toolkit.

```
# top -m io -o read
…
  PID USERNAME        VCSW  IVCSW   READ  WRITE  FAULT  TOTAL PERCENT COMMAND
35765 www              931     85     48      0      0     48  10.50% nginx
35766 www              410     66     32      0      0     32   7.00% nginx
 4994 root           66139   3971     10      0      0     10   2.19% nfsd
35248 www             1113     39      6    143      0    149  32.60% nginx
 3975 mysql            205     28      1    145      0    146  31.95% mysqld
```

zfs-stats

On Solaris, ZFS uses a system called kstat to publish various statistics about what is happening internally in ZFS. On FreeBSD those stats are published via the sysctl interface. The `sysutils/zfs-stats` package can summarize those statistics in a more human-readable way, logically grouping them together.

```
# zfs-stats -A
------------------------------------------------------------------------
ZFS Subsystem Report                            Thu Oct 19 03:50:59 2017
------------------------------------------------------------------------

ARC Summary: (HEALTHY)
        Memory Throttle Count:                  0

ARC Misc:
        Deleted:                                92.50m
        Recycle Misses:                         0
        Mutex Misses:                           14.31k
        Evict Skips:                            6.79k

ARC Size:                               99.87%  31.92   GiB
        Target Size: (Adaptive)         100.00% 31.96   GiB
        Min Size (Hard Limit):          12.50%  4.00    GiB
        Max Size (High Water):          8:1     31.96   GiB

ARC Size Breakdown:
        Recently Used Cache Size:       98.18%  31.38   GiB
        Frequently Used Cache Size:     1.82%   594.11  MiB

ARC Hash Breakdown:
        Elements Max:                           690.77k
        Elements Current:               75.55%  521.90k
        Collisions:                             5.89m
        Chain Max:                              4
        Chains:                                 8.13k
```

MEMORY THROTTLE

The more important indicator of problems is the "Memory Throttle Count." This is the number of times that the ZFS ARC has had to reduce its memory usage because of demands elsewhere in the system. You might consider setting the maximum size of the ARC **(vfs.zfs.arc_max)** to a value that makes ZFS coexist with your other workloads better. The output also shows a breakdown of the ARC usage by MRU (Most Recently Used) and MFU (Most Frequently Used). This gives you some insight into how the cache is adapting to the workload.

zfs-mon

The **sysutils/zfs-stats** package also includes a second tool, zfs-mon, which looks at how a subset of the kstats are changing over time. This can provide useful insight into how the requests are being broken down, and how the various caching layers in ZFS are being used. The stats break down the performance of the ARC, L2ARC, the filesystem prefetch, and the device prefetching code. It also breaks down data vs metadata operations. By default, ZFS limits the amount of cache available for metadata to 25% of the max ARC size. If the total storage capacity is very large—and most operations impact only the metadata of the files, not the content—increasing the amount of the ARC that can be used for metadata can actually increase performance, since otherwise the ARC may be 3/4s full of content that will not be referenced again before it is replaced with other content.

```
# zfs-mon -a
ZFS real-time cache activity monitor
Seconds elapsed: 329

Cache hits and misses:
                                    1s    10s    60s    tot
                        ARC hits:  130    305    591   2875
                      ARC misses:   38    113     62    142
           ARC demand data hits:    95    236    539   2642
         ARC demand data misses:     2     51     29     18
       ARC demand metadata hits:    34     46     36    207
     ARC demand metadata misses:     0     36     17     90
         ARC prefetch data hits:     1     17     13     23
       ARC prefetch data misses:    36     26     16     33
     ARC prefetch metadata hits:     0      6      3      2
   ARC prefetch metadata misses:     0      1      0      0
                    ZFETCH hits:    28     69     50    189
                  ZFETCH misses: 18639  17450  18029  23507
             VDEV prefetch hits:     0      3      1      5
           VDEV prefetch misses:     0     39     12     18

Cache efficiency percentage:
                           10s    60s    tot
                   ARC:  72.97  90.51  95.29
       ARC demand data:  82.23  94.89  99.32
   ARC demand metadata:  56.10  67.92  69.70
     ARC prefetch data:  39.53  44.83  41.07
ARC prefetch metadata:  85.71 100.00 100.00
                ZFETCH:   0.39   0.28   0.80
         VDEV prefetch:   7.14   7.69  21.74
```

As you can see, the ARC cache hit ratio varies quite a lot over short intervals, but in the 5-1/2 minutes this tool ran, the overall average was a 95.29% hit ratio.

GEOM STATS

gstat is an interactive tool that pulls statistics from the FreeBSD GEOM subsystem. It can be a useful window into what is happening with the underlying storage. For each GEOM object (there may be many that represent a single device, or a partition or other subdivision of a device), the depth of the queue, total operations per second, read operations per second, read kilobytes per second, milliseconds per read operation, and all the same again for write operations are printed. Then a synthesized '% busy' number is calculated, a best guess only, and can often be seen exceeding 100%. There are additional operation types (delete for TRIM/UNMAP etc., and flush) that can be shown with additional flags. If the sum of the read and write IOPS per second is less than the value in the ops/s column, it is likely that these other operations are happening as well.

```
# gstat -f da..\?$
 L(q) ops/s    r/s   kBps   ms/r    w/s   kBps  ms/w  %busy Name
  0      0      0      0    0.0      0      0   0.0    0.0| ada0
  0      0      0      0    0.0      0      0   0.0    0.0| ada1
  5    733      0      0    0.0    733  62244  10.5   88.8| ada2
  7    883      0      0    0.0    883  62443   7.1   85.9| ada3
  7    961      0      0    0.0    961  62539   5.2   61.6| ada4
  0    960      0      0    0.0    960  63425   8.6   73.4| ada5
  7   1047      0      0    0.0   1047  65006   5.5   79.1| da0
 10   1078      0      0    0.0   1078  60751   5.9   81.4| da1
```

DTRACE TOOLKIT

DTrace is a very powerful tool designed to allow you to safely inspect and debug the running system, while having very minimal impact on performance when not debugging. DTrace scripts vary in complexity, from simple one liners to interactive tools.

A simple example of a DTrace one liner:

```
# dtrace -n 'syscall::read:entry { @bytes[execname] = sum(arg2); }'
```

This creates an aggregation of the 3rd argument (they are numbered from 0) of the read system call, by the calling application's name. Run this for a few seconds, then hit control+c to stop it. It will then print out a list of every application that called read, and the total number of bytes that were read. Now it is obvious which application was causing all of the reads from disk.

You don't have to write your own DTrace scripts; the OpenDTrace project maintains a collection of cross-platform scripts that you can download and use. These serve as a great starting point that can be modified to answer the questions you want to ask; https://github.com/opendtrace/toolkit

To look at how much data is being written in each transaction group, or how long each transaction group is taking to sync to disk, check out these DTrace examples by Adam Leventhal: http://dtrace.org/blogs/ahl/2014/08/31/openzfs-tuning/

CONTINUOUS PERFORMANCE MONITORING

Understanding the cause of performance problems first requires having something to compare the new measurements and observations against. Is the current level of operations per second typical? Or is it much higher or lower than expected. In order to make sense of the cache hit ratio, you need to know what it is when the system is NOT having problems. In order to have this information, and to be able to make sense of it, you need to be continuously recording the metrics that you will want to compare the current state of the system against. Collecting, storing, and graphing these metrics in a useful way is the key to being able to quickly diagnose issues and detect problems early.

Disks can be very ungentlemanly when they fail. Rather than loudly dying and going offline completely, they often misbehave. One of the first signs that a disk is beginning to fail can be greatly increased read

and write latencies. Consumer grade disks often retry internally many times before returning a read error. The operating system might then helpfully ask the drive to retry a few more times, each of those repeated commands resulting in a series of additional internal retries. Because of this, operating systems will often have rather high timeouts while waiting for commands to complete, with defaults on the order of 30 seconds per command and 5 retries. A single failed read can thus hold up the entire system for 2-1/2 minutes.

ZFS KSTATS

ZFS presents an impressive number of stats and counters via the kstat interface. On FreeBSD, this is currently exposed via the kstats.zfs sysctl mibs.

One of the advantages of ZFS is the ARC (Adaptive Replacement Cache), which provides better cache hit ratios than a standard LRU (Least Recently Used) cache. Looking at the various stats about the ARC can provide insight into what is happening with a system.

- `kstat.zfs.misc.arcstats.c_max` — The target maximum size of the ARC.
- `kstat.zfs.misc.arcstats.c_min` — The target minimum size of the ARC. The ARC will not shrink below this size, although it can be adjusted with the `vfs.zfs.arc_min` sysctl.
- `kstat.zfs.misc.arcstats.size` — The current size of the ARC; if this is less than the maximum, your system has either not had enough activity to fill the ARC, or memory pressure from other processes has caused the ARC to shrink.
- `kstat.zfs.misc.arcstats.c` — The current target size of the ARC. If the current size of the ARC is less than this value, the ARC will try to grow.
- `kstat.zfs.misc.arcstats.p` — How much of the ARC is to be used for the MRU list; the remainder is the target for the MFU list. This value will adjust dynamically based on workload. A lower value suggests frequent access to the same blocks, where a higher value suggests a more varied workload.
- `kstat.zfs.misc.arcstats.arc_meta_used` — The amount of the ARC used to store metadata rather than user data. If this value has reached `vfs.zfs.arc_meta_limit` (which defaults to 25% of `vfs.zfs.arc_max`), then consider raising or lowering the fraction of the ARC used for metadata. Caching more metadata will increase the speed of directory scans and other operations, at the cost of decreasing the amount of user data that can be cached.

SNMP

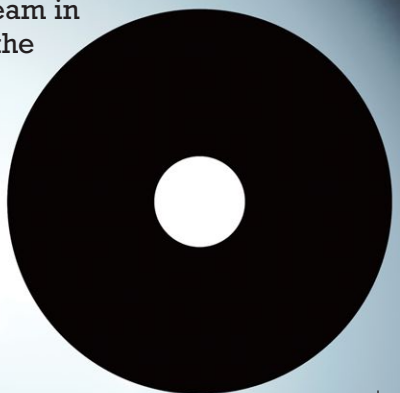
net-snmpd provides a number of useful counters like total IOPS per device and bytes read and written. These can be used to create graphs to provide some historical perspective when looking for performance problems. Is the IOPS load twice what it normally is? That might be your problem. If the number of IOPS is down, but the workload is higher, something might be causing one or more devices to perform suboptimally.

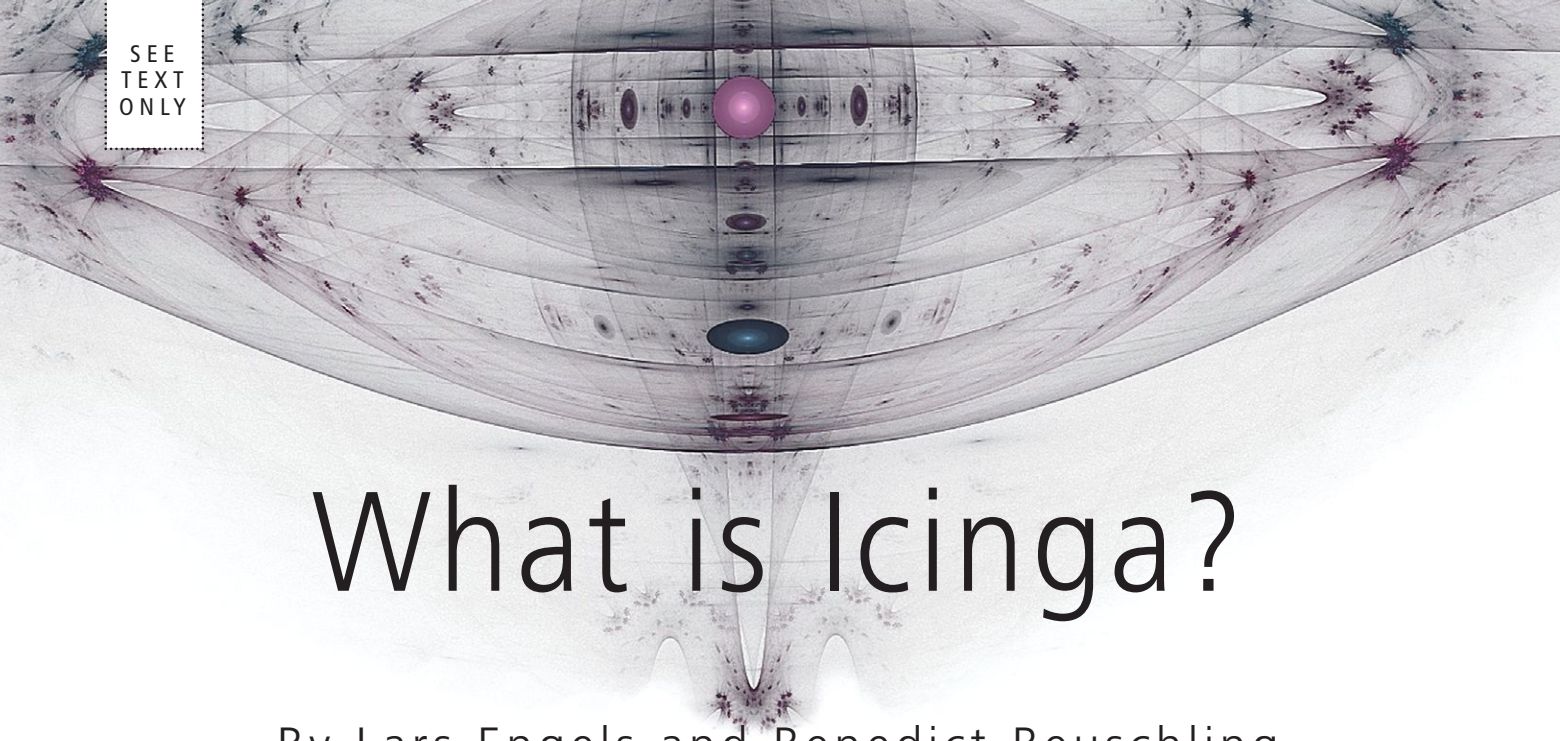
OTHER TOOLS

There are many different solutions for monitoring, measuring, and recording statistics from a system. Some you might wish to investigate include:

- Zabbix — An advanced monitoring suite with some predefined probes for ZFS.
- Collectd — A metrics collection daemon that can be used with a number of different backends.
- Grafana — A graphing and analytics tool for time series data that can make sense of metrics gathered by applications such as collectd.
- OSQuery — An operating system instrumentation framework for analyzing the live and historical metrics of a system using a familiar structured query language. ●

ALLAN JUDE is VP of operations at ScaleEngine Inc., a global HTTP and Video Streaming Content Distribution Network, where he makes extensive use of ZFS on FreeBSD. Allan is a FreeBSD src and doc committer, and was elected to the FreeBSD core team in summer 2016. He is also the host of the weekly video podcast BSDNow.tv (with Benedict Reuschling), and coauthor of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* with Michael W Lucas.





What is Icinga?

By Lars Engels and Benedict Reuschling

During the early ages of network monitoring, there was a monitoring software called Netsaint, which was later renamed to Nagios for legal reasons. While Nagios was very successful in the open-source world, patches from the community were often rejected or added very slowly. In 2009, members of the Nagios community created a fork that was named Icinga (Zulu for: it "examines"). Since the release of Icinga 2.0 in 2014, it is no longer a fork of Nagios, but a complete rewrite with new features like distributed monitoring, HA-Clustering, a REST API, and more. Icinga can reuse all Nagios monitoring plugins, while providing a modern web interface and powerful extensions to its core functionality.

Installation on FreeBSD/ZFS

In this article, we will walk through a setup of Icinga and Icinga Web 2 on FreeBSD to monitor hosts. The configuration data, events, and user authentication will be stored in a PostgreSQL database (MySQL is also available by default). NGINX will serve the web pages for Icinga Web 2. It is assumed that there is a FreeBSD base installation already installed and that it is connected to the network, able to download packages and reach other hosts via `ping` and `ssh`. This setup has been tested on a Raspberry Pi 3, as well as on a regular server without requiring any special hardware or tuning parameters.

The Icinga documentation (https://docs.icinga.com) has an excellent description of the necessary Icinga installation steps and even provides FreeBSD-specific instructions (when paths are different, for example). A FreeBSD port/package is also available and already includes some steps that users of other Unix distributions must still perform.

The easiest way to install Icinga 2 on FreeBSD is to use the package net-mgmt/icinga2. This guide uses # in front of commands that should be performed by the root user and `$` for a non-root user. Alternatively, `sudo` can be used to temporarily elevate privileges and to execute commands as another user.

Installing Icinga 2

We begin by installing a couple of packages required for the setup (see Box 1). Icinga 2 will install the basic monitoring components. Icinga Web 2 is an optional component to display events and check results in a browser-based dashboard. PostgreSQL 9.5 is the database server being used

Note

This guide concentrates on getting Icinga 2 up and running, and won't focus on performance, SSL setup, and other security options (other than passwords). This is left as an exercise for the reader, once you have become more familiar with Icinga 2 and its features.

by the package at the time of this writing. Although the setup of NGINX is described here, Apache 2 users can run Icinga Web 2 just as well.

The packages ImageMagick-nox11 and pecl-imagick are installed separately, as they are not pulled in as dependencies by default. When we start to configure Icinga Web 2, it will complain about these missing components when they are not installed. They are needed to create graphs in the PDF output when generating reports. Without them, the report PDFs will have stats and metrics in textual form only. Other dependencies like PHP, which is used by Icinga Web 2, will be pulled in automatically by the packages listed here.

Box 1
```
# pkg install icinga2 icingaweb2 postgresql95-server nginx ImageMagick-nox11 pecl-imagick
```

After all components have been installed, we add entries using `sysrc(8)` (see Box 2) to `/etc/rc.conf` so that these components will be automatically started upon reboot.

Box 2
```
# sysrc icinga2_enable=yes
# sysrc postgresql_enable=yes
```

Setting up PostgreSQL

Before we begin setting up the PostgreSQL database and create users for Icinga, we create a ZFS dataset to hold the data (UFS users can just use `mkdir` to follow along). Replace `monitor` with the name of your pool in the following examples:

Box 3
```
# zfs create -o mountpoint=/usr/local/pgsql/data monitor/pgdata
# zfs set compression=lz4 monitor/pgdata
# zfs set recordsize=8k monitor/pgdata
# zfs set logbias=throughput monitor/pgdata
# zfs set redundant_metadata=most monitor/pgdata
# zfs set primarycache=metadata monitor/pgdata
# chown pgsql:pgsql /usr/local/pgsql/data
```

The more systems are being monitored, the more writes to the database will be performed, so we tune ZFS accordingly. With these settings, the database transactions will not only be compressed on disk, but will also honor the metadata settings the database uses, so ZFS will not assume to know any better than PostgreSQL when it comes to writes to disk.

Now it is time to log in as the `pgsql` user and create the database cluster by running `initdb` with the `data` directory (dataset) that we just created:

Box 4
```
# su pgsql
$ cd
$ initdb -D ./data -E UTF8
$ pg_ctl start -D ./data
```

We create a new database user `icinga`, a database with the same name, and assign the `icinga` user as the owner. Provide a password for the icinga user when asked. It is required later when we set up Icinga Web 2, so make sure to not forget it.

Box 5
```
$ createuser -dPrs icinga
$ createdb -O icinga -E UTF8 icinga
```

The `pg_hba.conf` file was generated when `initdb` was executed and controls which user can access the database. To allow the icinga user to connect only locally to the database, edit the `pg_hba.conf` file in `data` and add the following lines:

Box 6
```
local      icinga      icinga                                  md5
host       icinga      icinga          127.0.0.1/32            md5
host       icinga      icinga          ::1/128                 md5
```

The icinga database needs a couple of tables, indices, and references before it can record the monitoring data from the hosts. This schema is provided by the port/package and located in `/usr/local/share/icinga2-ido-pgsql/schema/pgsql.sql`. The `psql` command interpreter is used to execute the SQL-script directly:

Box 7
```
$ psql -U icinga -d icinga < /usr/local/share/icinga2-ido-pgsql/schema/pgsql.sql
```

After the script was executed successfully, the database-specific configuration steps are done. Log out of the `pgsql` user before continuing. Icinga connects to the database via IDO (`Icinga Data Out to Database`) and Icinga needs to know about this. Icinga 2 has a plugin-like interface that can enable and disable features. For IDO to work with the PostgreSQL database, we need to enable the `ido-pgsql` feature. Another feature that is needed is called `command`. Afterwards, Icinga 2 is started for the first time.

Box 8
```
# icinga2 feature enable ido-pgsql
# icinga2 feature enable command
# service icinga2 start
```

Icinga is basically running now and will perform checks for the local machine. Having a nice dashboard to get an overview of any reports and any incidents is much better, so we configure Icinga Web 2 and the NGINX webserver next.

Configuring NGINX for Icinga Web 2

Icinga Web 2 is running on PHP, and we will use PHP FPM to make it work with NGINX. First, we enable `php-fpm` and `nginx` to start when the system is rebooted:

Box 9
```
# sysrc php_fpm_enable=yes
# sysrc nginx_enable=yes
```

Then, we use a couple of `sed` commands to configure the `php-fpm` configuration file. The first line lets it listen to the local domain socket and the three lines below it uncomment owner, group, and mode options to use the ones provided by FreeBSD (i.e., the `www` user and the default permissions for the socket).

Box 10
```
# sed -i '' "s/listen\ =\ 127.0.0.1:9000/listen\ =\ \/var\/run\/php5-
fpm.sock/" /usr/local/etc/php-fpm.conf
# sed -i '' "s/;listen.owner/listen.owner/" /usr/local/etc/php-fpm.conf
# sed -i '' "s/;listen.group/listen.group/" /usr/local/etc/php-fpm.conf
# sed -i '' "s/;listen.mode/listen.mode/" /usr/local/etc/php-fpm.conf
```

Icinga Web 2 provides an example file for `nginx.conf` in `/usr/local/share/examples/icingaweb2/nginx/icingaweb2.conf`. We take the relevant portions and put them before the `location / {` line in `/usr/local/etc/nginx/nginx.conf`:

Box 11
```
location ~ ^/icingaweb2/index\.php(.*)$ {
  # fastcgi_pass 127.0.0.1:9000;
  fastcgi_pass unix:/var/run/php5-fpm.sock;
  fastcgi_index index.php;
  include fastcgi_params;
  fastcgi_param SCRIPT_FILENAME /usr/local/www/icingaweb2/public/index.php;
  fastcgi_param ICINGAWEB_CONFIGDIR /usr/local/etc/icingaweb2;
  fastcgi_param REMOTE_USER $remote_user;
}
location ~ ^/icingaweb2(.+)? {
  alias /usr/local/www/icingaweb2/public;
  index index.php;
  try_files $1 $uri $uri/ /icingaweb2/index.php$is_args$args;
}
```

PHP needs to be configured as well. Fortunately, FreeBSD provides a configuration file for PHP with reasonable settings for production use, aptly named `php.ini-production`. We copy this file to the location of the `php.ini` file:

Box 12
```
# cp /usr/local/etc/php.ini-production /usr/local/etc/php.ini
```

One line still needs to be added to let PHP know which time zone it is in. Substitute the example below with the time zone in which your monitoring server is located.

Box 13
```
# echo "date.timezone = Europe/Berlin" >> /usr/local/etc/php.ini
```

Time to start both PHP-FPM and the NGINX webserver:

Box 14
```
# service nginx start
# service php-fpm start
```

If everything works, open a browser and point it to the following URL to begin the Icinga Web 2 configuration: http://local.domain.or.ip/icingaweb2/setup.

If something went wrong, retrace the steps above and have a look at the log files located in `/var/log/icinga2` for any clues about what went wrong. Log rotation examples for newsyslog can be found under `/usr/local/share/examples/icinga2/newsyslog`.

Now, let's configure the Icinga Web 2 dashboard.

Configuring Icinga Web 2

The first welcome screen asks for the setup token. This is done to prevent someone from accidentally stumbling onto the freshly installed Icinga server and misconfiguring it (remember this could be the evil Internet). To create the necessary setup token, run the following commands:

Box 15
```
# /usr/local/www/icingaweb2/bin/icingacli setup token create \
  --config=/usr/local/etc/icingaweb2
# chown -R www:www /usr/local/etc/icingaweb2
```

Copy the setup token that is being echoed on the screen and enter it in the setup token field. Click Next to continue. The next screen will show what kind of modules the installation has detected. Make sure that at least "Monitoring" is checked, and click Next. To make sure the Icinga installation runs with all the required components (mostly PHP modules), Icinga will provide an overview page of what it detected in the local installation. When following this guide, most of these fields should be green, meaning that the module is present and can be used. The "Linux Platform" field can be safely ignored, as the software runs just as well on FreeBSD. Click Next once more.

This screen asks how users will authenticate against Icinga Web. We'll use the PostgreSQL database we set up earlier, but LDAP works just as well for corporate environments. After selecting "database," we'll continue to the next page where we'll enter our connection information for it. Make sure to select "PostgreSQL" as the database type; localhost and the port should already be correct. Enter "`icinga`" as the database name. The user and password is the same we provided in the postgresql setup above. Enter `UTF-8` as the encoding. You can decide whether you want a persistent connection to the database, which is usually a good idea when using the web interface often. You can validate the connection to see whether everything works before continuing on to the next page.

This page asks which authentication backend to use, and should already provide a default entry. More authentication backends can be added later in Icinga Web 2, so just continue here. Enter the credentials (username and password) for the administrative user account on Icinga Web 2 in the next screen. Make sure to remember the password before continuing.

The following screen deals with debugging (whether stacktraces should be created, which are user-visible), where settings are stored, and how logging should be done (logging type and log level). Logging to a separate file is a good idea if syslog shouldn't be cluttered with Icinga 2 messages. Run `mkdir /var/log/icingaweb2` followed by `chown www:www /var/log/icingaweb2` when switching to

"`file`" so that the web interface is able to write its logs there.

The next page summarizes all your settings, and when you are satisfied with these, go to the next page to configure to monitoring component. The backend is IDO, which we already used for the PostgreSQL database, so we just continue on from here. We want to store all monitoring information in PostgreSQL, so we have to enter the same connection information we used earlier. Confirm that the settings are correct by validating them before hitting the Next button.

The command transport is important when multiple instances of Icinga are being used, but in our case, we accept the local file transport settings and continue on. No need to make changes to the defaults provided in the variables to be protected. Another summary page follows before we can finally finish the setup for Icinga Web 2 and log in for the first time. Use the administrative account created earlier in the setup to get to the monitoring dashboard.

Monitoring Hosts and Services

Monitoring is a complex topic and Icinga has many different ways to monitor a remote system. The Icinga 2 documentation has plenty of examples available.

The easiest way is via simple ping checks, which we'll configure for an example host.

Open `/usr/local/etc/icinga2/conf.d/hosts.conf` and add an entry like this (Box 16):

Box 16

```
object Host "Example" {
  import "generic-host"
  address = "IP-ADDRESS"
}
```

Replace "`Example`" and "`IP-ADDRESS`" with the hostname and IP address, respectively. After saving and exiting the file, let Icinga 2 check its config (`# service icinga2 configcheck`) and restart the icinga2 service. The Icinga Web 2 interface should now display a new pending host in the dashboard, and a short time later, the host should be regularly checked by pings. In case the host is down, Icinga 2 will display a warning in the dashboard and remove it once the host can be reached again.

Icinga Can Do Even More...

Icinga can be extended with more functionality beyond monitoring machines and their services. For example, in complex environments, where there are a lot of objects being monitored in a hierarchical fashion, a business process module can help visualize them. This way, many thousands of cloud machines that form the basis for a service can be viewed from a high-level dashboard. In case there is an alert, the module allows you to drill down to the exact machine that is triggering the alert. [https://github.com/Icinga/icingaweb2-module-businessprocess]

Icinga Director makes it easy to handle Icinga 2 configurations. By allowing users the flexibility to create their own objects by "point & click," while at the same time completely automating their datacenter, sysadmins can be sure their monitoring solution grows with higher demands for their infrastructure. [https://github.com/Icinga/icingaweb2-module-director]

Another module adds a generic trouble ticket system (TTS) functionality to Icinga 2. It allows the replacement of ticket patterns in Icinga Web 2 with links to a trouble ticket system (TTS). It defines a ticket hook that can be used by the core monitoring module and others for acknowledgements, downtimes, and comments. [https://github.com/Icinga/icingaweb2-module-genericts]

Lastly, system administrators dealing with systems distributed in many geographical locations can get a better overview with the map module for Icinga Web 2. It uses OpenStreetMap data to display host objects and their status on a map. Multiple hosts at the same location are clustered together. There is a custom host action to locate a specific host on the map. Clicking on a host marker will open a pop-up that will display that host services' current status. [https://github.com/nbuchwitz/icingaweb2-module-map]

Also, there's a module that integrates Grafana graphs into Icinga Web 2 so you can display your check's performance data in an attractive way. [https://github.com/Mikesch-mp/icingaweb2-module-grafana]

All of these additional modules are available as FreeBSD ports/packages. Make sure to study the Icinga 2 documentation to monitor your hosts and services in an effective way. •

LARS ENGELS has been a FreeBSD ports committer for 10 years and is also loosely involved in the Icinga project.

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he began mentoring others to become FreeBSD committers. He is a proctor for the BSD Certification Group and is currently serving as vice president for the FreeBSD Foundation.

SEE TEXT ONLY

By Poul-Henning Kamp

STDDEV or: "It Didn't Happen!"

Many, many years ago, back when Dilbert cartoons were still funny, I tried to do what can best be described as "nano-benchmarking." The basic idea was to boot two hardware-identical FreeBSD computers in disk-less configuration, run them for several days, and measure how proposed patches to the FreeBSD kernel code improved or worsened performance.

It soon became obvious that the quartz crystals on the motherboards were better at measuring temperature than time, and that limited my precision to, at best, three significant digits.

Being a "time-nut," I replaced the crystals with an external signal from a high-quality ovenized, quartz crystal, which, again, was steered using GPS signals, from a state-of-the-art, "Motorola Oncore," timing GPS receiver. That got me to eight significant digits.

.....Or did it?

That's when I voluntarily broke out the statisics textbook and started to appreciate statistics as a tool.

Statistics is not any normal person's favorite branch of mathematics; it involves a lot of number-crunching and it gives fuzzy answers:

> 65% of all Americans believe that frozen pizza will never be any good, and there's nothing science can do about it. Source: Widgery & Associates. Margin of error within 9%. From a telephone survey of 204 Americans, Spring 1993 [1].

But even the most cursory study of the developments from the previous century will reveal that it is statistics, all the way from Erlang's formulas for telephone system capacity, over Shewhart's quality control of the manufacturing of telephone handsets, to Googles PageRank, and the current "Big Data" bubble, which I trust some future historian will chronicle in a book called "Covariances Gone Wild."

Statistics is powerful, but hard. My favorite piece of overlooked statistics—an article which fixed the odds of global warming to at least 100,000 against one (in 1992, back when Windows 3.1 came out)—

took me a full week to comprehend [2].

Because statistics is hard work, most people just skip it entirely, and, therefore, almost all claims about performance improvements that came across the FreeBSD mailing lists, including some of my own, were just unsubstantiated and anecdotal claims.

Around the time of FreeBSD 5, I decided to do something about that, but what?

Pointing people at a heavy-duty statistics package such as 'R' would be a nonstarter: I had tried it myself and given up on it.

What the project needed was a "Software Tool": A program that did one thing well, and without a lot of work, mental or otherwise.

I wrote a ~500-line C-program, which read one number per line from one or two input files, plotted the data in a cheesy little ASCII-art plot, printed the basic statistical measures of each input file underneath the plot: minimum, maximum, average, median, and standard deviation.

If given two input files, it would calculate the Student's T test, and report if there were more than a 95% chance that the two data sets differed in a meaningful way:

```
$ ministat iguana.txt chameleon.txt
x iguana.txt
+ chameleon.txt
+--------------------------------------------------------------------------+
|x         *  x                  *     +                    +    x       +|
| |_____________M____|___A_________________M__A_____________________|       |
+--------------------------------------------------------------------------+
    N           Min           Max        Median           Avg        Stddev
x   7            50           750           200           300    238.04761
+   5           150           930           500           540    299.08193
No difference proven at 95% confidence
```

Ministat is an incredibly primitive tool. The ASCII-plot looks horrible; it has no scales and is barely good enough to spot outliers, bifurcations, and other major data trouble, but it has the big advantage that you can email it as plain text.

Student's T is a relatively new breakthrough in statistics, a mere 100 years old [3], and while it is

not the statistically optimal tool for all circumstances, it is very robust with respect to the input data, and one does not need a lot of skill to interpret its yes/no result. Which is why I committed ministat to FreeBSD with the message:

> If your benchmarks are not significant at the 95% confidence level, we don't want to hear about it.

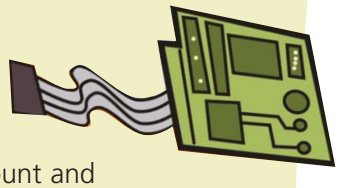
The example data files that go with the source code are from "The Cartoon Guide to Statisics"[4], a competent introductory statistics textbook, written as a comic full of "dad-jokes," by the inimitable Larry Gonick [5].

Ministat was not an overnight success; it came with the full stigma of being "statistics," but after a couple of years of reminding people of its existence, it had become the de facto way to argue performance improvements in FreeBSD, and I saw it regularly in BSDcan presentations.

I think OpenBSD was the first place it appeared outside FreeBSD, but today there are also packages for Linux and ports of ministat written in at least Python, Ruby, and Erlang.

And every so often, strangers offer me a beer, because, of course, ministat had to be beerware licensed: Student's T was discovered by William Gosset, an employee of the Guinness Breweries [6].

PHK's Tips for Good Benchmarks



- Run in single user mode. Cron(8) and all the other daemons only add noise.
- Make sure the CPU always runs at the same clock-speed. (BIOS settings etc.; AC-power, etc.)
- If syslog events are generated, run syslogd with an empty syslogd.conf; otherwise, do not run it.
- Minimize disk-I/O; avoid it entirely if you can.
- Don't mount filesystems you do not need.
- Mount filesystems read-only if possible.
- Newfs your R/W test filesystem and populate it from a tar or dump file; then unmount and mount it again before every run.
- Use malloc backed or preloaded MD(4) partitions if you can. When writing, SSDs are more unpredictable than rotating platter disks.
- Remove all unnecessary device drivers from the kernel. For instance, if you don't need USB for the test, don't put USB in the kernel. Drivers that attach often have timeouts ticking away.
- Unconfigure hardware you don't use. Detach disk with atacontrol and camcontrol if you do not use them for the test.
- Do not connect or configure the network unless you are testing it.
- Do not run NTPD unless your test takes days to complete.
- Minimize output to serial or VGA consoles. Running output into files on a ramdisk causes less jitter.
- Make sure your test is long enough, but not too long. If your test is too short, timestamping is a problem. If it is too long, temperature changes can ruin it. Rule of thumb: more than a minute, less than an hour.
- Try to keep the temperature as stable as possible around the machine. Temperature changes affect both quartz crystals, CPU thermal management, and seek algorithms in disk drives.
- Run at least 3, but preferably many more, tests of the "before" and "after" code, and analyze the results with ministat.

(continues next page)

- **PHK's usual test-plan:**

a[1] b[1]
Do a sanity-check.

a[2] b[2] a[3] b[3]
Run ministat on a[1:3] vs b[1:3]
Do not despair if there is no difference.

a[4] a[5] b[4] b[5] a[6] a[7]
b[6] b[7] a[8] a[9] b[8] b[9]

Run ministat on a[4,6,8] vs a[5,7,9] and
on b[4,6,8] vs b[5,7,9]
If there is a difference, stop and think.

Run ministat on a[1:9] vs b[1:9]
If the result is solid, you can stop here.

a[10] a[11] a[12] b[10] b[11] b[12]
a[13] a[14] a[15] b[13] b[14] b[15]
a[16] a[17] a[18] b[16] b[17] b[18]

Run ministat on a[1:3] vs a[16:18] and
on b[1:3] vs b[16:18]
If there is a difference, stop and think.

Run ministat on a[1:18] vs b[1:18]
This is your final result; it is unlikely
to improve much.

[1] Michael Moore: Adventures in a TV Nation, p. 9. ISBN 0-330-41914-5

[2] Statistics of Extreme Events with Application to Climate Change. https://fas.org/irp/agency/dod/jason/statistics.pdf

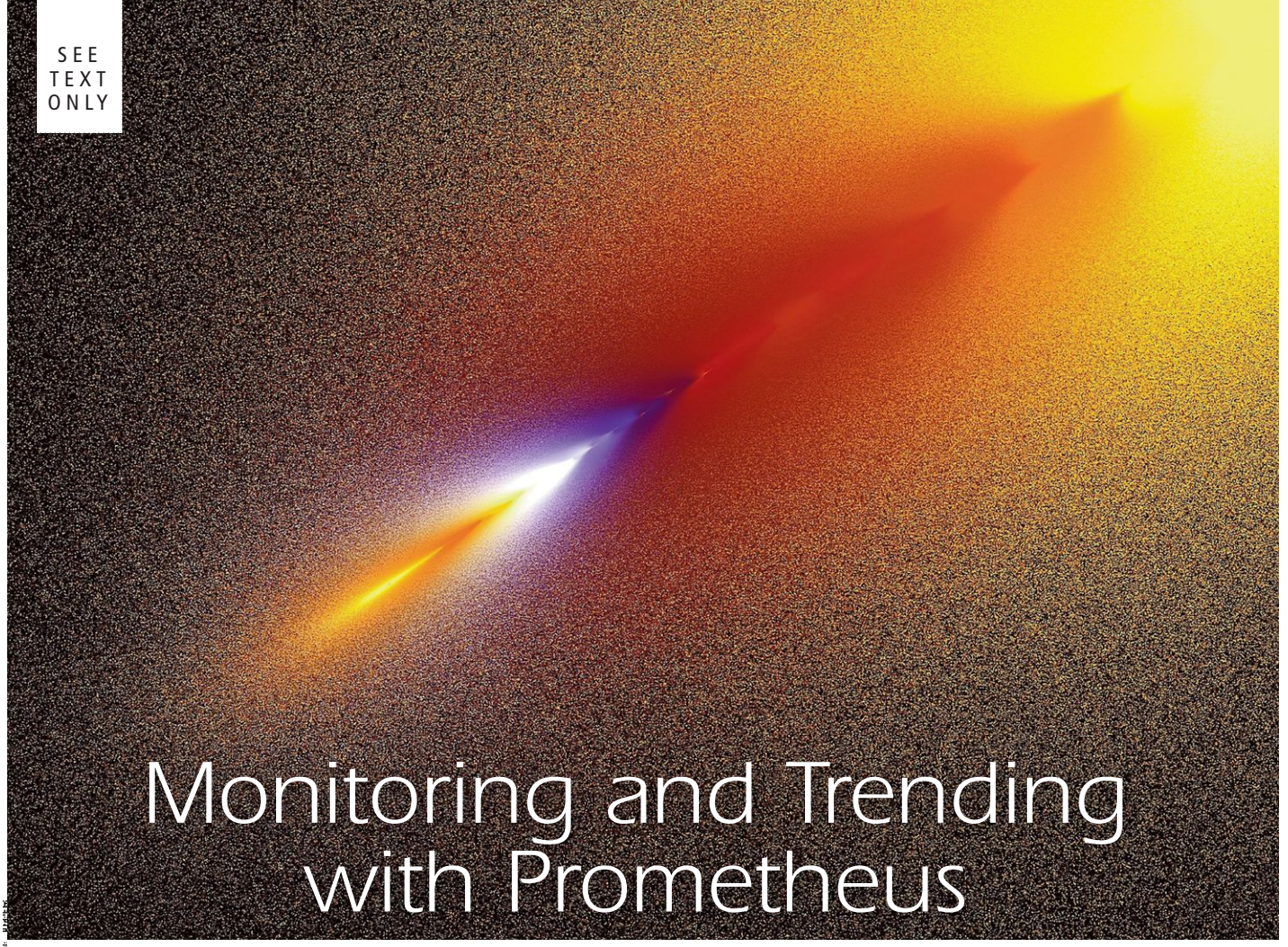
[3] Walter A. Shewhart, the founder of statistical quality-control, positively gushed about Student's T when he heard about it in 1926. https://archive.org/details/bstj5-2-308

[4] http://www.larrygonick.com/titles/science/the-cartoon-guide-to-statistics/

[5] http://www.larrygonick.com/ Highly recommend all of his books, but in particular the history series.

[6] https://en.wikipedia.org/wiki/William_Sealy_Gosset

Poul-Henning Kamp is phk@FreeBSD.org, and he used to be one of the most active kernel programmers in the project. These days his main project is the Varnish HTTP-cache, but his laptop still runs FreeBSD-current.



Monitoring and Trending with Prometheus

BY ED SCHOUTEN

Founded in 2012 by software engineers at SoundCloud, Prometheus is a monitoring system whose design has been inspired by Borgmon, the system that keeps an eye on jobs running on Google's internal Borg cluster. Here, we'll discuss how Prometheus works, following a hands-on approach. In the process, we'll also look at some other tools that are often used in conjunction with Prometheus, such as Grafana and some of the metrics exporters. At the end of the article, we're also going to look at an automation tool that we are developing and will release as open-source software in the near future, called Promenade.

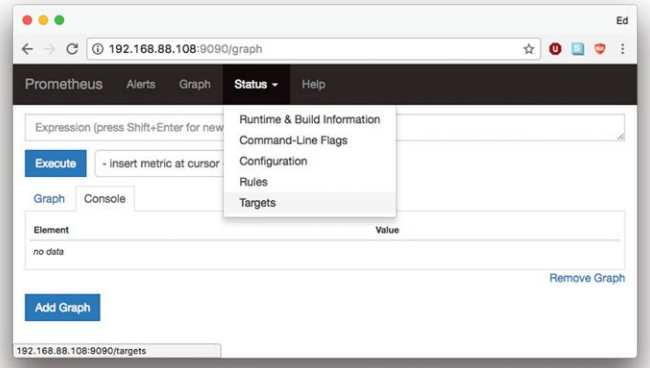
Getting Started with Prometheus

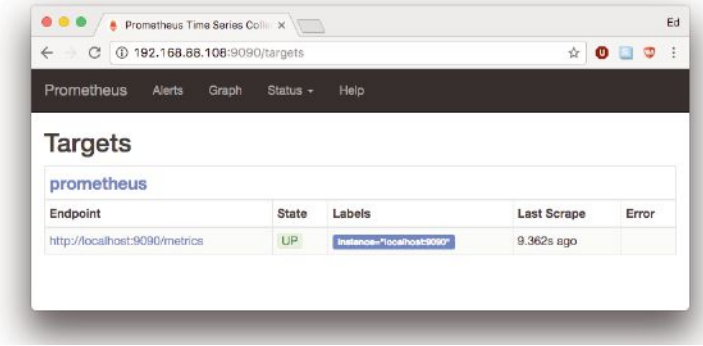
Though Prometheus can already be installed very easily using Go's build utility (`go get`), there is also a version that is packaged by the FreeBSD ports collection. One advantage of installing this package is that it ships with an `rc(8)` script that allows us to easily run it as a daemon. Prometheus should be up and running after running the commands below:

```
$ pkg install prometheus
…
New packages to be INSTALLED:
      prometheus: 1.7.1
…
Proceed with this action? [y/N]: y
…
$ sudo /usr/local/etc/rc.d/prometheus forcestart
Starting prometheus.
```

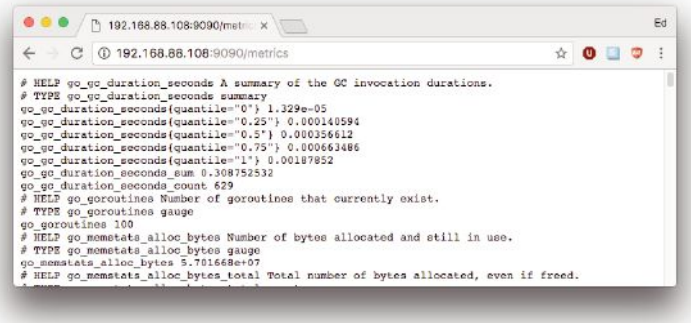
Starting Prometheus

By default, Prometheus binds a HTTP server on port 9090. If we point our browser to this server, we are presented with a page on which we can explore the data stored in Prometheus. As we've just fired up this instance, there will, of course, be little data to explore. Let's leave this page alone for now and head over to the "targets" page (2 screenshots below).





The targets page shows us which endpoints are being monitored by Prometheus, and already reveals something interesting. This instance of Prometheus has been configured to monitor itself. Prometheus is a white box monitoring system, which means that it can store metrics that are reported by targets themselves, as opposed to only measuring externally visible factors (e.g., TCP and HTTP health checks). By clicking on the HTTP link in the leftmost column, we can take a look at the raw metrics generated by the Prometheus server: stats related to the Go runtime's garbage collecting, threading, HTTP handling, and metrics storage (screenshot right).



Prometheus's format for serving metrics over the network is rather simple. Each metric is placed on a single line of a HTTP response and has a numerical, 64-bits floating-point value. Metrics are uniquely identified by a name and an optional set of labels placed between curly brackets (key-value pairs). Labels allow a program to return multiple metrics of the same name. For example, a web server could use labels to return HTTP statistics per registered virtual host or per HTTP error code class (e.g., latency of just the HTTP 200s for "www.freebsd.org").

To distinguish between identically named metrics returned by separate targets, Prometheus attaches additional labels while ingesting, such as `job` and `instance`. These labels contain values that uniquely identify the endpoint. The values for these labels are shown on Prometheus's targets page. At Kumina, we use this mechanism to attach custom labels relevant to our environment, such as physical location (data center name), system ownership (customer name), and support contract (24/7 or office hours only). These labels can then be used as part of queries and alert conditions.

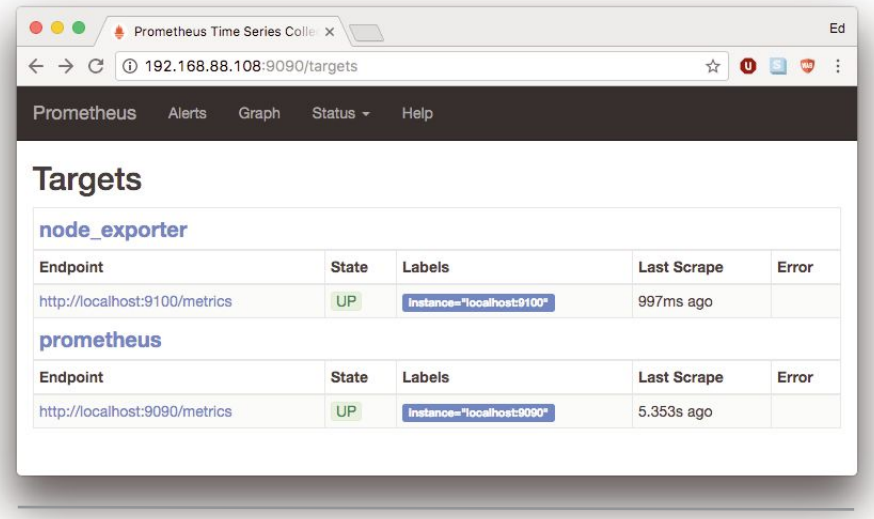
On its own, Prometheus is not capable of obtaining any operating system metrics, such as CPU load, disk usage, and network I/O. It is a tool that is capable only of ingesting metrics over HTTP and indexing them. System-level metrics are instead provided by a tool called the node exporter. The node exporter is nothing more than a web server that, when visited, extracts kernel-level state through `/dev`, `sysctl`, `libkvm`, etc. and returns it in Prometheus's metrics format. Installing the node exporter on FreeBSD is quite easy:

```
$ sudo pkg install node_exporter
…
$ sudo /usr/local/etc/rc.d/node_exporter forcestart
Starting node_exporter.
```

Once started, we want to extend Prometheus's configuration to scrape the node exporter as well. By default, the node exporter listens on port 9100.

```
$ sudo vim /usr/local/etc/prometheus.yml
<!-- Add the following entry under "scrape_configs": -->
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
$ sudo /usr/local/etc/rc.d/prometheus onerestart
Stopping prometheus.
Starting prometheus.
```

After restarting Prometheus and refreshing the targets page, we can see that it now scrapes two targets, which is what is expected (right):



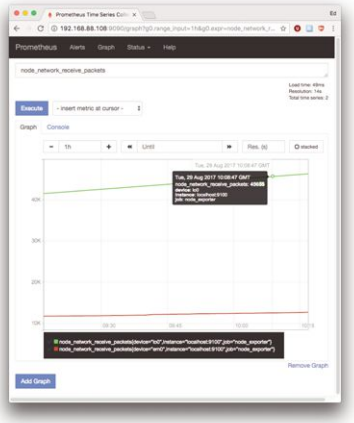
In addition to the node exporter, there are many other targets we could at this point configure. There exist exporters for most commonly used services (e.g., MySQL, Nginx, Java JMX) that convert metrics from their native format and serve them over HTTP. The Prometheus black box exporter can perform ICMP, DNS, TCP, HTTP, and SSH checks and report availability and latency. FreeBSD 12.x ships with `prometheus_sysctl_exporter(8)`, a tool that can serve values of arbitrary sysctls. Some of these exporters are officially maintained by the Prometheus project, while there are many others that are maintained by the community.

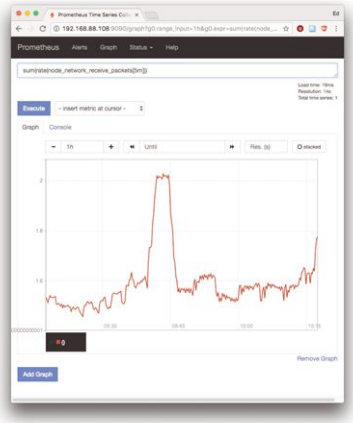
At Kumina, we have developed exporters for services including Dovecot, PHP-FPM, libvirt, OpenVPN, and Postfix. We have also designed a simple network traffic accounting daemon based on libpcap that exports per-address statistics, called Promacct. All of these tools are available on our company's GitHub page (https://github.com/kumina).

If you want to use Prometheus to obtain metrics from software that is being developed in-house, there is no need to make use of separate metrics exporter processes. The Prometheus project provides client libraries for various programming languages (Go, Java, Python, Ruby, etc.) that make it possible to directly annotate your code with metrics objects. For languages like Python and Java, these libraries also provide convenient function decorators to automatically count function invocations and create histograms of their running times.

PromQL: Prometheus's Query Language

After letting Prometheus gather metrics for a couple of hours, we can head over to the graphing page to explore Prometheus's dataset. Let's start off by graphing a single metric that is generated by the node exporter, `node_network_receive_packets`. As the name suggests, this metric corresponds with the number of network packets received by the system's network interfaces. This expression produces a graph with two lines on my system: one for the loopback interface (`device="lo0"`) and one for the physical interface (`device="em0"`).





If we wanted to plot metrics only for certain hosts or network interfaces, we could append filters to the end of the expression. For example, appending `{device!~"lo[0-9]+"}` to our query would remove metrics for loopback devices by making use of negative regular expression matching. `{datacenter="frankfurt"}` would only give us results for systems in a single data center, if such a label were to exist.

Graphing the number of network packets directly doesn't seem to show us anything useful; it only renders two diagonal lines. The reason for this is that the node exporter reports the number of network packets as a counter that accumulates over time. Only when the system running the node exporter reboots (or an integer overflow of the counter occurs), it resets to zero. By letting an exporter use cumulative counters, it may safely be scraped by multiple Prometheus servers. It also makes exporters oblivious of the scraping interval configured on the Prometheus server. Even if the Prometheus server needs to reduce the number of scrapes due to high load, no packets will remain unmeasured.

In order to convert the number of packets into something meaningful, we will first need to compute its derivative. This can be accomplished by performing two operations on the metric. First of all, instead of querying for the metric's scalar value, we're going to request vectors of consecutive samples, called range vectors. We can then use the `rate()` function to turn those range vectors back into scalars, representing the rate of increase of a range vector per second. The size of the range vectors used will determine the smoothness of the resulting graph. Five-minute range vectors are good for spotting brief bursts of traffic, whereas one-hour range vectors can visualize daily traffic curves. At Kumina, we even use 7-day and 31-day range vectors for long-term network and CPU capacity planning. When changing the graph expression to `rate(node_network_receive_packets[5m])`, we get a graph that shows the number of received packets per second, using a 5-minute rate computation.

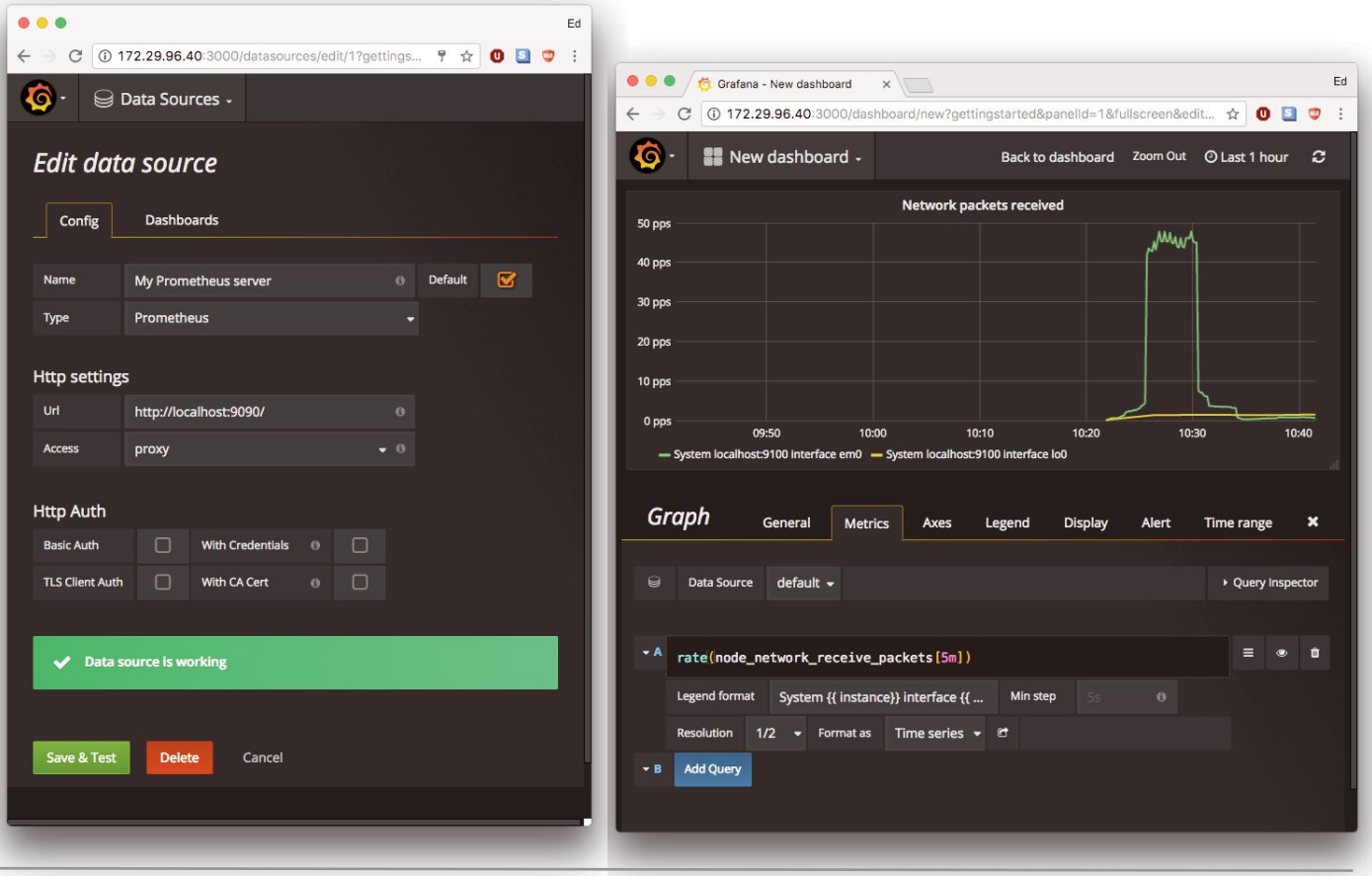
Prometheus's query syntax also supports aggregation operations with which we can reduce the number of metrics graphed. For example, the query `sum(rate(node_network_receive_packets[5m])) by (instance)` will compute bandwidth per server, as opposed to showing bandwidth per network interface. The query `topk(10, rate(node_network_receive_packets[5m])) by (datacenter)` will limit the output to just the top 10 per data center.

Creating Dashboards with Grafana

Though Prometheus's built-in web application is all right for running queries interactively, it is often desired to design dashboards that show graphs in an organized way. Prometheus used to ship with a feature called Promdash that allowed the server to serve templated HTML files. This feature has been removed in the meantime, as there are now various third-party dashboard tools that can interface with Prometheus directly, the most prominent one being Grafana. Getting Grafana to work on FreeBSD essentially follows the same recipe as what we've seen before:

```
$ sudo pkg install grafana4
…
$ sudo /usr/local/etc/rc.d/grafana forcestart
Starting grafana.
```

With its default settings, Grafana will spawn a web server that listens on TCP port 3000. After pointing our browser to it and logging in with the default credentials (username "admin", password "admin"), we are presented with Grafana's home screen. The first thing we'd want to do is click on "Add data source" to configure the address of the Prometheus server that Grafana should use to fetch data—in our case, http://localhost:9090/.



Once completed, we can click on the next button on the home screen, titled "Create your first dashboard". We then get presented with an empty dashboard page on which we can place panels, such as graphs, tables, heat maps, and lists. In the case of Prometheus, using graphs makes sense most of the time. When creating graphs, we can use the same query syntax as we've used before. By pressing the save icon at the top of the page, the dashboard gets saved on the Grafana server.

Depending on the specifications of your hardware, you may notice that dashboards will take a longer time to render as the number of graphs increase and graph queries become more complex. Executing many complex queries at the same time may cause a significant load on the Prometheus server. To solve this, Prometheus has the option to pre-compute complex queries while scraping and store its result under a different name, using recording rules. As a rule of thumb, you should use recording rules for graph queries as soon as they are more complex than simple selection expressions.

```
$ sudo vim /usr/local/etc/prometheus-rules.yml
<!-- Add the following contents: -->
my_recording_rule =
     sum(rate(node_network_receive_packets[5m]))
     by (instance)
$ sudo vim /usr/local/etc/prometheus.yml
<!-- Add the following entry under "rule_files". -->
  - prometheus-rules.yml
$ sudo /usr/local/etc/rc.d/prometheus onerestart
```

The commands above show how Prometheus can be configured to make use of recording rules. In this example, we're adding a recording rule called `my_recording_rule`, which may from now on be used in graph queries. Recording rules can be named arbitrarily, but to improve readability, it makes sense to apply some best practices. The Prometheus documentation has an article on the naming scheme suggested by the developers. For the recording rule above, the Prometheus documentation suggests it should be named `instance:node_network_receive_packets:rate5m`.

Alerting

In addition to graphing, Prometheus is also capable of generating alerts based on metric values. Alerts can be configured by declaring alerting rules that may be placed in the same file as recording rules. Below is an example of what an alerting rule looks like:

```
ALERT TargetFailedToScrape
   IF up == 0
   FOR 15m
   LABELS { severity = "page" }
   ANNOTATIONS {
      summary = "Instance {{ $labels.instance }} is down!",
      playbook_url = "http://intranet.company.com/...",
      ...
   }
```

This alert will trigger if a metric called "up" has the value zero for at least 15 minutes. The "up" metric is created by Prometheus implicitly to denote whether it has been able to scrape a target successfully. Labels that are part of the metric used in the alerting expression will also get attached to the alert itself. These labels are useful for formatting user-friendly alert messages, but also to create "silences", patterns for alerts that should be suppressed temporarily (e.g., due to planned maintenance). Prometheus will show all registered alerting rules and their state on its "Alerts" page.

To keep its design simple, the Prometheus server only supports a single mechanism for announcing active alerts, namely by sending REST calls to some other service. The Prometheus project provides a separate daemon called Alertmanager that can process these REST calls, generate email, SMS, and Slack messages, and manage silences. The URL that Prometheus uses to perform REST calls can be configured through the `--alertmanager.url` command line flag. It may also be necessary to set the `--web.external-url` flag to the public URL of the Prometheus server, so that Alertmanager can add clickable links to its alert messages that point back to Prometheus.

Federation: a Hierarchy of Prometheus Servers

There are scenarios in which it is undesirable to use a single Prometheus server to collect metrics for all of your infrastructure. The number of targets to scrape and metrics to ingest may become too large for a single Prometheus instance. Targets may also be spread out geographically, meaning there is not a single location from which all targets can be scraped reliably. To solve this, Prometheus servers can receive HTTP GET requests on `/federate?match[]=...` to print selected stored metrics in its own format, allowing them to be ingested by other servers.

In practice, you see that this mechanism is often used to introduce hierarchy. A service that is distributed across multiple data centers may have one Prometheus server per location, scraping only the systems nearby. These Prometheus servers then use recording rules to aggregate key metrics for the entire data center, which are then scraped by a global Prometheus server. With root causing problems, one can first investigate the dashboards provided by the global Prometheus instance to determine which data centers are affected. Access to metrics on a system level can be obtained by switching to the appropriate local instance.

An advantage of such a hierarchical setup is that it may reduce disk space usage significantly. Local Prometheus instances can be configured to use volatile storage and have a short retention period (days), whereas the global instance may make use of persistent storage and have a very long retention period (years), as it only needs to hold on to a tiny fraction of the data. Storing data for multiple years is useful for long-term capacity planning.

Creating Dashboards from Python with Promenade

At Kumina, we've noticed that managing recording rules on a large scale may be quite challenging. Refactoring Prometheus configuration files always has a risk of breaking existing Grafana dashboards, especially because dashboards are stored in a web-based tool, as opposed to being stored in a version control system next to the Prometheus configuration files. We're therefore working on implementing a utility to configure Prometheus and Grafana installations programmatically, called Promenade. With

Promenade, you can write Python classes that declare Grafana dashboards as follows:

```
class UnboundMetricsBuilder:
    def construct(self):
        yield Dashboard(
             title='Unbound',
             rows=[
                DashboardRow(title='Queries', graphs=self._row_queries()),
                ...
             ])
    def _row_queries(self):
        yield Graph(
           title='Query rate per data center',
           queries=[
              GraphQuery(
                 expression=Sum(
                      expression=Rate(
                         expression=Metric('unbound_queries_total'),
                         duration=datetime.timedelta(minutes=5)),
                      by={'datacenter'}),
                 format='Data center %(datacenter)s')
           ],
           unit=Unit.OPERATIONS_PER_SECOND,
           stacking=Stacking.STACKED,
           width=WIDTH_FULL // 2)
     yield Graph(
           ...
```

An interesting aspect of Promenade is that it can take a hierarchy of Prometheus servers into account, which can also be declared as Python code. In the code below, we first declare a DAG (Directed Acyclic Graph) of how labels on metrics are related to each other (e.g., all metrics with the same "datacenter" label value will also have the same "country" label value). We then declare a hierarchy of Prometheus server objects for which we want to generate configuration files.

```
label_implications = LabelImplications({
    ('instance', 'customer'),
    ('instance', 'rack'),
    ('rack', 'datacenter'),
    ('datacenter', 'country'),
    ...
})

local = ScrapingRecordingServer(label_implications)
global = CompositeRecordingServer(
    label_implications, 'datacenter', {local}, ...)
```

When combining the Python code for our dashboard with this hierarchy, Promenade will automatically create the following recording rule on the local Prometheus instance:

```
datacenter:unbound_queries:rate5m =
    sum(rate(unbound_queries_total[5m]))
    by (datacenter, country)
```

It will also generate a configuration file for the global Prometheus instance, so that it will scrape `datacenter:unbound_queries:rate5m` from all of the local instances, which allows Grafana to access them.

At Kumina, we are currently migrating all of our existing Prometheus and Grafana setups to be built on top of Promenade, which is why the design of Promenade is still being tweaked to meet our requirements. We are planning on releasing it on our company's GitHub page (https://github.com/kumina) as soon as the code has stabilized, so stay tuned!

Wrapping Up

We hope that this article gives a good impression of how easy it is to get started with using Prometheus to monitor your systems. At Kumina, we have been happy users of Prometheus for about a year now. It is a robust, flexible, and extensible monitoring system, having a healthy ecosystem of both developers and users. Within the next couple of months, we will see the release of Prometheus 2.0, which will include lots of new features. At Kumina, we are most excited about the redesign of the storage layer, which will allow us to collect millions of samples per minute on servers with commodity hardware.

Since 2016, the Prometheus team has organized an annual conference called PromCon (https://promcon.io/). This year it took place in Munich, Germany, on August 17–18. If you are interested in knowing more about some of the latest developments taking place in the Prometheus project, be sure to check out the conference's website, as recordings of all of the talks are publicly available. ●

ED SCHOUTEN is the lead software developer at Kumina, a managed services provider and consultancy firm based in Eindhoven, the Netherlands. Kumina provides companies with fully managed platforms and offers support, training, and consultancy for Prometheus and Kubernetes.
Feel free to visit our website https://kumina.nl/ or contact us at info@kumina.nl to tell us about your project or request more information about our offerings.

SEE TEXT ONLY

svn UPDATE

by Steven Kreuzer

The past few installments of svn update have all followed a theme, and I did my best to find recent commits to go along with the particular theme. Suffering from a case of writer's block, I decided to make this month's theme a hodgepodge of changes I find interesting, which, I guess, you could actually consider a theme. I hope you will enjoy this random selection of interesting commits as much as I have. Hopefully, when the new year rolls around, my brain will be in better shape. Here's looking forward to another incredible year of FreeBSD development!

Use estimated RTT for receive buffer auto resizing instead of timestamps. https://svnweb.freebsd.org/changeset/base/316676

Switched from using timestamps to RTT estimates when performing TCP receive buffer auto resizing, as not all hosts support / enable TCP timestamps. Disabled reset of receive buffer auto scaling when not in bulk receive mode, which gives an extra 20% performance increase.

The amazon-ssm-agent package is installed by default on EC2 AMI builds. https://svnweb.freebsd.org/changeset/base/325254

This makes it immediately available on instances that are running without Internet access (or that can't rely on firstboot_pkgs to install it for some other reason).

Support for compressed kernel dumps. https://svnweb.freebsd.org/changeset/base/324965

When using a kernel built with the GZIO config option, dumpon -z can be used to configure gzip compression using the in-kernel copy of zlib. This is useful on systems with large amounts of RAM, which require a correspondingly large dump device. Recovery of compressed dumps is also faster since fewer bytes need to be copied from the dump device.

Support for labeling md(4) devices. https://svnweb.freebsd.org/changeset/base/322969

This feature comes from the fact that we heavily rely on memory-backed md(4) in our build process. However, if the build goes haywire, the allocated resources (i.e., swap and memory-backed md(4)'s) need to be purged. It is extremely useful to have the ability to attach arbitrary labels to each of the virtual disks so that they can be identified and GC'ed if necessary.

Support for Intel Software Guard Extensions (Intel SGX). https://svnweb.freebsd.org/changeset/base/322574

Intel SGX allows management of isolated compartments "Enclaves" in user VA space. Enclaves memory is part of processor reserved memory (PRM) and is always encrypted. This allows protection of user application code and data from upper privilege levels including the OS kernel.

The lengths of geli passwords are now hidden during boot. https://svnweb.freebsd.org/changeset/base/322923

CloudABI compatibility has been synced against version 0.13. https://svnweb.freebsd.org/changeset/base/322885

With Flower (CloudABI's network connection daemon) becoming more complete, there is no longer any need for creating any unconnected sockets. Socket pairs in combination with file descriptor passing is all that is necessary, as that is what is used by Flower to pass network connections from the public Internet to listening processes.

Support for ZFS Channel Programs have been imported from OpenZFS. https://svnweb.freebsd.org/changeset/base/324163

ZFS channel programs (ZCP) add support for performing compound ZFS administrative actions via Lua scripts in a sandboxed environment (with time and memory limits). The initial commit includes both base support for running ZCP scripts, and a small initial library of API calls that support getting properties and listing, destroying, and promoting datasets.

Support for suspending and resuming a zpool scrub. https://svnweb.freebsd.org/changeset/base/323355

Scrubbing can be an I/O intensive operation and people have been asking for the ability to pause a scrub for a while. This allows one to preserve scrub progress while freeing up bandwidth for other I/O.

Support setting the colors of cursors for the VGA renderer. https://svnweb.freebsd.org/changeset/base/322878

Add a new tool for doing experiments with SDIO cards. https://svnweb.freebsd.org/changeset/base/320847

Add mmcnull, an emulated lightweight MMC controller. https://svnweb.freebsd.org/changeset/base/320845

This emulated device attaches to the ISA bus and registers itself as HBA supporting MMC/SD cards. This allows the development and testing of MMC XPT and MMC / SDIO peripheral drivers even in the VM such as bhyve.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retrocomputing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

SEE TEXT ONLY

new faces of FreeBSD

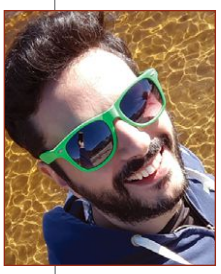
BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community. This past quarter was busy for new committers. In this issue's column, the spotlight is on Fedor Uporov, who received a src bit and Luca Pizzamiglio, who received a ports bit in August; Adriaan de Groot and Craig Leres, who each received a ports bit in September; Ilya Bakulin and Chuck Tuffli, who each received a src bit in September; and Yuri Victorovich, who received a ports bit in October.

Tell us a bit about yourself, your background, and your interests.

• Fedor: I received a master's degree in radio engineering nearly six years ago, and then started my career as a hardware designer. I had worked with different microcontrollers and FPGA, and then started to make my first efforts to program these, not as regular job tasks, but just as a hobby. Later, I found that my professional interests shifted from hardware development and PCB design to programming. And then I changed my job and became a programmer.

I have really simple interests. When I have free time and when I am not tired from the monitor and keyboard, I try to find time to work with open-source code or to learn something new. Otherwise, I prefer walking or travelling.

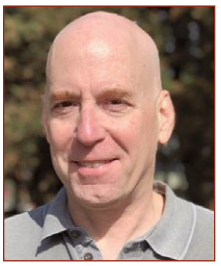


• Luca: I'm 39 years old, and I was born in Casalpusterlengo, a small city in Italy. I graduated from the Politecnico di Milano in computer science engineering in 2003. I was lucky enough to receive my first 8086 when I was 10, and since then I've been interested in computers. I worked as consultant for some years in Italy, working with FPGA and Linux. In 2009, I moved to Berlin, Germany, where I started to work on FreeBSD and learned to speak something close to German. Now I live in Dusseldorf, still working on FreeBSD at Trivago.

When not working with computers, I play volleyball and beach-volley, and have started a fencing class, which is really a lot of fun. I also play piano, and I sang for some years in a jazz choir.

• Adriaan: I grew up in Calgary, Canada, and did silly things on an Apple until my high school got some PC juniors. Then I spent a short time at the U of C, where I met men with long grey beards, wearing hiking boots, seated at vt100 terminals. There was some kind of UNIX running on the other end, and like a doofus, I wrote my own build system (make used tabs, which I disagreed with) and then my own editor (vi had command-mode, which I disagreed with). Shortly after that, I moved to the Netherlands to escape the consequences of that hubris, and ended up with an email address in 1990 and "adridg" as login—which has been my username in the 25 years since.

The Netherlands continued to be my home, where I received a PhD, had some kids, and enjoyed a varied career in the software industry—currently doing free software development full-time. If you give me an hour's notice, you can stop by for dinner: there's a fair chance of dinner being vegetarian, and I'll gladly cook for groups up to 40 people. I can play electric bass (badly), Kerbal Space program (badly), and a lawyer on TV (ditto). The lawyering thing is connected to my interest in licensing issues; for a while I carried a card that said, "I talk to lawyers so you don't have to." That's the card I showed to developers; the other side said, "I talk to engineers so you don't have to," for the other end of the conversation.



• **Craig:** When I was a little kid my father taught me how to solder, and over the years I built a lot of little electronic projects. When I was about 11, my father's wealthy friend bought a PDP-8/e, had the stock exchange wired into our house, and father wrote some code in assembly to watch and report on a couple stocks. Once his friend got bored with the project, I was able to use the PDP anytime I wanted. This is when I learned to program, and started spending more of my time playing with computers than with electronics.

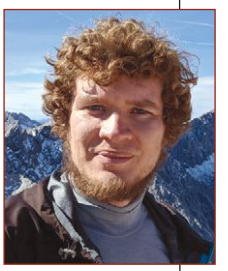
My first Unix account was on the Cory EECS 11/70 at UC Berkeley, while I was a student. Later, I became an unpaid volunteer with the Computer Systems Research Group (CSRG) at Berkeley and had root and source commit privileges for Berkeley Software Distribution (BSD). I was one of the maintainers of the termcap file and my name appears in this file on Unix and Linux systems that still ship with it. I coauthored /usr/bin/write when BSD was working on rewriting programs that were encumbered by the System V source license.

At that time, I was also working at the Lawrence Berkeley National Laboratory in the Network Research Group (NRG). Another group had a Vax 11/780 that ran BSD. It was an ARPANET node, and when I first got my account, it used the Network Control Program (NCP) protocol. The admins would sometimes boot the experimental TCP kernel after hours and eventually transitioned to it full-time. With guidance from my boss (Van Jacobson), I was introduced to Unix network drivers, and we added a watchdog to the token ring driver used to talk to a couple of VMS systems in the building. I was also involved with another fun project, where we installed a custom-built microwave link between LBL and UCB. It provided 230.4 Kb/s site-to-site communication.

Our own group had a Vax 11/780 that ran VAX/VMS. But we also had an early version of software called Eunice that provided an environment that would run BSD network device drivers on top of VMS. We actually tested a lot of the early NRG TCP enhancements on this platform. It was common for me to port the current BSD networking stack to Eunice while Van was out of town giving a paper somewhere.

In the mid-1980s, we started getting Sun workstations, and soon I had 3/50s on my desktops at work and home. We built our kernels from source (which was based on BSD at the time) and started using SunOS 3 to do network stack development. One memorable change was when Van noticed that mbuf clusters were 1K long, but Ethernet frames were ~1500 bytes. By doubling the size of a cluster (and making a few other tweaks), he was able to get 3/50s to transfer data at close to 10 Mb/s over ThickNet Ethernet. His testing would frequently crash our Vax/VMS system; while Ethernet uses exponential back-off, DECnet used linear back-off and VMS did not limit how much memory could be allocated to DECnet transmit retries and would panic when network utilization was unusually high.

My interests include NASCAR (I'm a part-time stock car racing school instructor), electronics (digital audio and especially anything Arduino based), science fiction, and, of course, programming.



• **Ilya:** I graduated from the Moscow Engineering Physics Institute in 2009, with the intention to work in the micro- and nanoelectronics industry, but quickly changed my direction to software development. I have since been working at different companies, first in Moscow, and then in Munich, Germany, all of them employing BSD systems to build their products. Currently I work as a site reliability engineer at Google in Munich, and hack on the FreeBSD kernel in my free time.

When I'm not behind my laptop's screen, I usually go to the mountains and spend time there hiking or skiing, either alone or with my wife and kids.

• **Chuck:** My name is Chuck Tuffli, and I currently live outside of Sacramento, California. I've been using computers for "a while" (post punch cards, but before floppy drives were popular), and had the fortune to stumble into the world of "software that talks to hardware." The majority of my recent experience is with storage-related hardware (Fibre Channel, SAS, NVMe), but my past is checkered with various video, graphics, and WiFi drivers. When not programming, I spend time with my family and our horses. Somewhat related, if you ever have the burning need to fix fences or irrigation, just let me know. I can hook you up.

• Yuri: I have an MS degree in physics. At the university, I specialized in theoretical physics, computed the anomalous magnetic moment of elementary particles, and did a lot of mind-boggling math. After the university, science remained my passion, but my interests also included computers and technology. I have worked as an engineer and later as an architect in a variety of industries: insurance, gaming, consumer electronics, optics, and electronic design automation (EDA), which develops software to design microchips. I've always gravitated toward algorithmically complex and science-related fields, since they tend to be the most challenging. I am also very interested in chemistry and biochemistry.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• Fedor: Nearly three years ago, I started working with the Mac OS X darwin kernel, mostly VFS and filesystems. I started to learn FreeBSD, because darwin VFS was branched from BSD some time ago. And it turned out that the BSD side is more readable and understandable than the darwin kernel. So, FreeBSD really helped me to manage OSX VFS in terms of my regular job tasks.

• Luca: The first time I heard about FreeBSD, I was at the university and my friends were completely fascinated by the user 'toor'. At that time, I was a Linux guy, Slackware was my distro. When I moved to Berlin, I had the pleasure of working with my first mentor, vwe@, unfortunately only for a short time. What I really like in FreeBSD is how smooth it is building every piece of software that is running on your machine. It makes it really easy to hack your system and to contribute by submitting patches. I'm still building everything that runs on my laptop!

My first work on FreeBSD was GELI and different legacy ways to provide passphrases. Currently, I work mainly on ports, but I've started to play with jails and ZFS. Moreover, my interest in DTrace is growing every day, and it's incredible what you can do with it.

• Adriaan: There's a FreeBSD 2.2.3 boxed set on my bookshelf. I probably ran that on my home PC, which had two vt100s connected. At the same time, I was developing little networked games on X11 on SunOS, and next to the 2.2.3 boxed set is a Red Hat Linux 4 manual, so I was probably just experimenting with whatever I could get my hands on.

Around 1998, I got into international free software development, contributing patches to KDE and becoming maintainer of KPilot, the Palm Pilot synchronization tool. Somehow, I ended up chatting with Lauri Watts, the documentation coordinator for KDE, and she ran FreeBSD and invited me over to the dark side. That must have been around 2001, and so I ran FreeBSD at home for a long time, and Linux at work, where I needed a (binary only) CMU LISP system to run a proof checker. After a few years, the KDE-FreeBSD community weakened and fell asleep—IRC nicks like tap, wca, lioux, and lofi fell silent on the KDE-FreeBSD channel.

I spent a few years working with OpenSolaris; by then I had a pretty clear idea of one thing I could do well: apply interesting tools to source code and find code quality issues—and then fix them. OpenSolaris had the Sun Studio compiler, which was free for a while. That had the primary incentive that it wasn't gcc, and, therefore, had a different view of what good C++ is. Applying that compiler to a huge codebase turned up tons of subtle gcc-isms, and plenty of Linux-and-BSD-isms that didn't apply to Solaris.

Somewhere in this history, SATA appeared on the horizon, and I spent a couple of months toying with Silicon Motion controllers and port replicators. I sent patches to the FreeBSD ATA subsystem maintainer—for a while I had possibly the only FreeBSD system that supported SiI3124 RAID with a port multiplier, although I didn't have six disks to attach to it. Shortly after that, the CAM subsystem took over, and I don't think anything ever happened with my code. The parts are still kicking around in my box-of-cruft. I should try them again someday.

Once OpenSolaris had been shuttered, I was looking for a platform to develop on again; with Clang on the horizon and FreeBSD's approach of nuts-and-bolts-included, I could return to applying interesting tools to KDE's codebase quickly. Most Linux distros have wandered off into developer-unfriendly territory, in my opinion, and, certainly for casual contributors, it's hard to start hacking on something in a clean-and-shiny distro. FreeBSD might have an ASCII art bootloader, but it gets you a system you can work with as a developer from the get-go.

Cue another few years of working with the renewed, reborn KDE-FreeBSD group, with tcberner@ and rakuco@ in the lead. I was the KDE guy doing FreeBSD, and they the FreeBSD guys

doing KDE, and it worked pretty well together. On the KDE side, we got FreeBSD added as a first-class citizen of the continuous integration system (snarkily, I'll add that one Linux distro was just canned for being notoriously difficult to keep running). Linuxisms are now routinely fixed, and upstream is generally more aware of "things could also be different." That may be a side effect of (KDE) also producing mobile and Windows packages, but I'll take it.

I think I was part of the organizing committee of a EuroBSDCon once, too, but I can't find any reliable references (neither is my memory).

- **Craig:** Having been a user, contributor, and fan of BSD, FreeBSD was the obvious next step when BSD was mothballed. Using FreeBSD always felt like using BSD to me. I also liked FreeBSD's reputation as a high-performance server platform.

By the mid-1990s we started running FreeBSD 2.X at LBL, first on our laptops and eventually on commodity PC hardware. I've run on FreeBSD desktops ever since then.

- **Ilya:** I set up my first FreeBSD server (based on FreeBSD 6.2-RELEASE i386) to serve as a router for the corporate network of the company I worked for. It replaced a big, noisy Windows-based server and was able to provide fast file storage and network access while running on an old PC. Back then, I was impressed with how effective and easy to maintain the system can be, and how it can squeeze everything out of aging hardware.

One of the companies I worked at built a FreeBSD/ARM-based telemetry solution, which is how I got in touch with running FreeBSD on embedded hardware. I have also been running FreeBSD on my laptop for a while.

- **Chuck:** Around FreeBSD 4.7, my employer had a driver development kit with a working driver for Red Hat Linux. At this time, there was a great deal of uncertainty around GPL enforcement and a worry that the Linux driver might force them to reveal proprietary information about their hardware. In order to hedge their bets, they decided it would be wise to also include a driver for a non-GPL, open-source operating system. After experimenting a little with the BSDs, FreeBSD became the reference design platform.

If the open-source license was all FreeBSD offered, I wouldn't still be here today. To paraphrase an old TV commercial, "come for the license, stay for the design." At the time, developing device drivers on FreeBSD was an order of magnitude better than Linux. Between the documentation, advanced storage features like SCSI target mode, and other well-thought-out and implemented subsystems, it was, and still is, an amazing development environment.

- **Yuri:** A friend mentioned FreeBSD in 1998, and I decided to try it. Since then, I've never gone back to Windows. What attracted me was the ability to dive into it, to learn how its various parts worked and interacted, and to hack it on all levels. FreeBSD is very well structured and cleanly designed. FreeBSD made it possible for me to learn how an operating system functions inside, an opportunity that no commercial OS can ever allow.

FreeBSD has the best software packaging system (ports) among all OSes. It allows any user to easily rebuild any package with a clear and easy procedure, anytime. As a matter of policy, absolutely all source downloads for ports are fingerprinted. This eliminates the possibility of MITM attacks, which no other system does. Other systems just never address some security problems in their packaging systems, and those are long solved in FreeBSD.

How did you end up becoming a committer?

- **Fedor:** I worked with the Linux ext file system in my job, so I tried to make some patches to improve the FreeBSD implementation. When my patches became relatively complex, I got the src commit bit.

- **Luca:** I've been attracted by open source as a concept since I was a student. I didn't feel very confident at that time, and felt that I wasn't skilled enough to be a valuable contributor. Then I started to work on FreeBSD, and thread support on devel/gdb had serious issues. I submitted some patches, and suddenly I became the port maintainer of gdb.

Joining Trivago, I had more opportunities to submit patches in the ports tree, so Lars (lme@) and Olivier (olivier@) proposed me as a ports committer.

- **Adriaan:** Early in 2017, tcberner@ and rakuco@ approached me saying that I should go for a ports committer bit; partly so that kde@ ports could be spread across a few more people, so that we would be more resilient—as we all have jobs and busy periods and real-life commitments too. After all, I've been using (and contributing to) FreeBSD off-and-on for at least 15 years, probably 20, and this feels like putting a ring on it.

- **Craig:** I've been a longtime contributor to Unix

in general, and FreeBSD in particular, submitting my first FreeBSD PR in 1999. I became a ports maintainer in 2008 (sysutils/lbl-hf), and created or picked up maintainership for additional ports over the following years. This is probably when I made becoming a committer a personal goal.

I ran into Kirk McKusick at a party last summer and mentioned my interest in being a committer. We know each other from when I was a CSRG volunteer, and he immediately offered to look into the process for me. Larry Rosenman and Matthew Seaman kindly volunteered to be my mentors. A short while later, I was invited to join FreeBSD as a ports committer.

- **Ilya:** Once I bought a somewhat expensive ARM-based home server, GlobalScale DreamPlug. It was possible to run FreeBSD on it; however, after setting it up, I found out that the onboard WiFi was connected to the system via SDIO bus. Back then, I didn't even know what SDIO was, and after reading some docs, I decided it would be a fun project to try to add support for SDIO bus. It turned out to be, well, a rabbit hole. After implementing the initial bits and discussing the problems I encountered with Warner Losh (imp@) and Adrian Chadd (adrian@) during EuroBSDCon 2013 in Malta, I started implementing a totally new SD/MMC stack based on the CAM framework. It ended up being an excellent project for learning a lot about FreeBSD kernel internals and for working with different people in the project to solve problems. Then Warner volunteered to be my mentor and let me improve FreeBSD MMC and SDIO support!

- **Chuck:** Michael Dexter's talk about diskctl got me curious about how to represent SMART buffers across the ATA, SCSI, and NVMe protocols. In the process, I wrote a little application that uses CAM(3) to send SMART commands to drives. But the CAM NVMe driver didn't have a couple of calls needed to make this work. Ten or so patches later to the NVMe driver, CAM, sgdd, and others, Ken Merry and Warner Losh got tired of committing code for me and asked if I'd like to be a committer.

- **Yuri:** I was discovering software that I found useful, and most times when this software wasn't available on FreeBSD, I was porting it. This way, I created about 200 ports, and in the process of doing this, I also fixed bugs in hundreds of other ports and submitted patches. Eventually, one committer offered for me to join the club. I agreed, but something unexpected came up and he didn't have time to be a mentor. Later, in late October, another committer, Tobias Berner, asked me again. I agreed, and that is how this happened!

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

- **Fedor:** I should say that the FreeBSD developers' community is really a friendly and fine group of people. I got this impression when I sent the self-introduction mail to the developers list. People may reply to your self-introduction mail in three to five days—it is really awesome. You get the feeling that your ideas and efforts are really needed for this project.

  Also, the FreeBSD mentorship is a really great way to help people pass their introduction phase and become committers. Here I should say thanks to Pedro Guffini (@pfg), who took responsibility for being my mentor. It has been a real pleasure for me to work with him.

- **Luca:** The knowledge that you can find in the FreeBSD community is smashing. Moreover, I was really welcomed by all developers. I've already made some small mistakes, and other port committers were so kind to let me learn from my errors, instead of just correcting them. I've been involved in constructive discussions from the beginning, and not just passively accepting the decision of senior contributors.

  To become a FreeBSD committer, the community has to know that you are a good contributor. So, contribute with patches, learn how to improve the quality of your contribution, and be visible on IRC and mailing lists. It's a process that needs some time, but it's also very rewarding.

- **Adriaan:** To some extent, things haven't changed much: I still run the same system, still spend most of my time on ports and pushing stuff into the KDE-FreeBSD ports github tree. I've picked up a couple of ports myself, for the heck of it (e.g., sayonara, because I want something slightly more capable than mpg123, but not much).

  For newcomers, I'd say "keep doing the things you love." And, since I've made my share of mistakes in my first months, "mind your p's and q's." Things can be very exacting in ports land, and it's natural to miss some little details at some point. Your mentors are there to help, but they're human too. The first "RE: svn commit" message you get

from mat@ pointing out an omission is a rite of passage, and just a reminder that we're all keeping an eye out to create the best systems we can.

- **Craig:** It's been great fun getting to see behind the curtain, gaining access to internal resources, learning the procedures, and seeing how the overall project fits together in greater detail. Everyone I've interacted with has been helpful and polite. It was very satisfying when I made my first "production" commit (as opposed to the commits related to becoming a committer). And being a committer is clearly the most efficient way to maintain a port, both for the maintainer/committer and other members of the FreeBSD Project.

  I don't know if my path to becoming a committer is typical, but I suspect it is a good model to follow. File PRs, become maintainer of several ports, and learn how the ports system works; read the documentation (obviously), but also /usr/ports/Mk makefiles. Look at existing ports to pick up tricks and optimal methods for making a port function well.

- **Ilya:** I had been contributing to the FreeBSD Project long before I became a committer—I maintained some ports, attended conferences, talked with people, and discussed ways of solving all sorts of problems. What you see pretty often is that when the system lacks certain features that many users need, people just go ahead and stuff gets implemented!

  What is important to understand is that this is an open-source project—so don't expect people to solve your problems and write code for you, and don't rely on any timelines until the feature really lands in the tree. Instead, try to research, debug and implement yourself, and come back with patches and ideas, and build a better system for everyone.

  A future FreeBSD developer is active in the mailing lists, goes to the conferences and actively participates in the discussions, and eventually finds a project that is fun to work on and is useful for other people in the community. Eventually their commit bit will find them!

- **Chuck:** FreeBSD has been a very welcoming community to me as both a contributor and committer. As for advice, my life is more often cited as the "cautionary tale" than in the "how to" category. That said, just go build stuff and be open to feedback from the community. The depth and expertise available here is a true gift and will make you better if you let it.

- **Yuri:** I have always had good experiences with FreeBSD, and can't imagine using any other system. I like the technology and I like the people. FreeBSD people are very friendly, skilled, and intelligent. FreeBSD committers are among the world's best computer experts. I was really touched by many warm welcome replies to my introduction email after I joined the committers community.

  I have only had a commit bit for two weeks now; it is probably too early to offer any advice. Just don't hesitate. It's well worth it! ●

DRU LAVIGNE is a doc committer for the FreeBSD Project and Chair of the BSD Certification Group.

conference REPORT

by Benedict Reuschling

EuroBSDcon 2017 Trip Report

EuroBSDcon in Paris, France, was a much-anticipated event. Throughout the year, at every BSD gathering, big and small, I heard people talking about it and making plans to get there. Even months before the conference, it was being billed as the biggest event so far, drawing attendees from all over the world. For me, it was an opportunity to give my first tutorial at EuroBSDcon, and it was also my first trip to France.

Arrival

On Tuesday, I flew into Paris, and soon found myself at dinner with Allan Jude and Niklas Zeising, who had come in with friends a few days earlier—really, it's Paris! Later that evening, Ed Maste joined us at an Indian restaurant where we did a lot of catching up.

The next day, before the board meeting, Ed Maste and I met up with Deb Goodkin from the FreeBSD Foundation for lunch in a nearby so-very-French bistro: sunshine, bistro chairs and tables on a small plaza with local people around us also enjoying the day. After updates on various projects, some paperwork, the requisite financials, and making plans for 2018 regarding sponsorship for conferences and events we'd like to attend, most of the members went back to their hotel rooms.

> "This is exactly the type of event where newcomers from the community get to show what they've done and get feedback and encouragement from developers, which will lead to further collaboration and code contributions." *—Benedict Reuschling*

Deb, however, accompanied me as I took advantage of my only window in which to do some sightseeing. We took a stroll toward Notre-Dame Cathedral, impressive outside, as well as inside. Mass was beginning and songs echoed through the centuries-old structure while sunlight mixed with candlelight. Dinner at an excellent Italian restaurant, with George Neville-Neil and others, was filled with conversation about FreeBSD and much more.

devsummit Day 1

The next day we walked to the Mozilla office, a beautiful old building rich with architectural ornaments juxtaposed with modern technology. Mozilla was the host of our two-day FreeBSD developer summit. I had helped organize the devsummit, putting together the schedule and asking people to submit working group proposals and talks. After Baptiste, one of the organizers of the event, and his assistants had registered everyone for the conference, we had breakfast (my second of the day!), which allowed people to meet and talk before the formal opening of the devsummit.

I opened the devsummit with something never done before. The person who could answer the most questions about France that I had prepared in Kahoot.com would receive a Mogics Power Bagel. Kahoot is a web application that people can connect with on their smartphone or laptop to answer a question that is currently projected on the screen. The trick is not only to be correct, but also to be faster than anyone else. After each round, the current high score is displayed, and participants also get feedback on their devices. It's a fun and interactive way of finding out what people know, with an application that is very popular in educational circles. After the first few rounds, it became clear that no one could beat Dag-Erling Smørgrav, and he won the prized Power Bagel. Look for this game at future events!

Next, we began the work-in-progress session. I had asked people to submit small 7-minute talks about their current work. The session was time-limited to keep people focused, but also to get them interested enough to continue the discussion during the breaks. First-time attendees could easily get to know other developers and what they are working on. We had a talk about a patch for GELI that allows attachment of multiple providers if they all use the same passphrase and keyfiles. Colin Percival gave a breakdown on how much time the kernel spends in certain parts when booting, and suggested ideas on how to shorten the boot time. He aims to reduce the kernel init time from roughly 11.5 seconds to around one second. Olivier Cochard-Labbe showed us his work on increasing FreeBSD packet-forwarding performance and the challenges associated with that. He illustrated it with flame graphs and charts plotting the linear reference numbers against various yet-to-be-committed patches and optimizations.

After the lunch break, Deb Goodkin provided an update on the FreeBSD Foundation, including current development projects and efforts to promote FreeBSD around the world. Scott Lamons, FreeBSD Foundation program manager, and Glenn Weinberg, of Intel, advised us about the current state of the partnership

we have with Intel. The main goals discussed were to foster the collaboration between the Foundation, Intel, and the FreeBSD community, and to improve the support of Intel products in FreeBSD. The rest of the afternoon, the microphone was open. From time to time, people would step to the podium and give a short impromptu presentation or ask for feedback on an issue they were working on. Later that day, we had an organized dinner, which was great for meeting and talking to even more people who were at the tutorials or had just arrived.

devsummit Day 2

The second day of the devsummit began with a talk by Ilya Bakulin on the status of the CAM-based MMC/SDIO stack. This is important work because many embedded boards use this interface to connect the various WiFi modules to it. So, with this work in the tree, it allows FreeBSD to access WiFi on many boards that currently have to use external WiFi via USB, for example. Next, core secretary Matthew Seaman and fellow members did a core team question-and-answer session. This provided us an opportunity to listen to feedback from developers on the work that the current core team is doing. We mentioned the FreeBSD Community Process again for those people who had not been to BSDCan this year where it had been introduced to a larger audience. Allan Jude and I also reflected a bit about our first-time experience being on core, and we encouraged people to run for core in next year's election.

After a coffee break, Scott Long from Netflix gave a talk about future work to provide FreeBSD with better non-uniform memory access (NUMA) support. A handful of people are involved in this, and the attendees agreed that it is an important area for development. I stepped out briefly to welcome Christian Schwarz. Although still a student about to start his bachelor thesis, he had already put in a lot of work on creating a ZFS replication solution called zrepl. I was impressed by this and had invited him to the devsummit to present his project which was well received: not only were people impressed by the work, but also the quality of his presentation. It brought a smile to my face when people suggested that he should submit his talk to AsiaBSDcon and BSDCan next year. This is exactly the type of event where newcomers from the community get to show what they've done and get feedback and encouragement from developers, which will lead to further collaboration and code contributions.

I had to leave the devsummit for my Ansible tutorial, but I heard that the rest of the devsummit had two working groups: one on sandboxing (Capsicum, Casper, and friends) and another on continuous integration in FreeBSD.

When I arrived at the tutorial venue, I met Michael Lucas, who had given his "Core Concepts of ZFS" tutorial that morning. I had submitted an updated "Managing BSD Systems with Ansible" tutorial to EuroBSDcon, originally given at AsiaBSDcon earlier in the year. Back then, there was already a lot of interest in the tutorial, and even more people turned up at EuroBSDcon—there was not an empty seat! I enjoy the interactive nature of a tutorial, as there is often someone in the audience who can provide more details or alternative approaches. This turns the session into a learning experience for everyone, and by the end, even the tutorial host has learned new bits of information. During the break, people tried out the examples or asked for the slides for later study. Feedback was good, so I'm considering submitting the tutorial as Version 3 to other BSD conferences (BSDCan 2018 comes to mind).

That night found us at yet another great and interesting restaurant, one with a cellar-like interior with an arched roof. And, of course, being the night before the conference, even more people were there. I noticed Brendan Gregg, whose keynote I was very much anticipating. A fair number of stories, quips, and other conversations were had over food and drink, after which we all returned to our hotels.

The Conference Day 1

The conference had everything you would expect from a good BSD conference: keynotes, three parallel tracks, breaks in regular intervals (a bigger one for lunch), breakout rooms, a recording crew and streaming, and, of course, a good hallway track. The conference took place in a Rosicrucian temple, with plenty of room for the over 300 attendees. Egyptian statues were around us and hieroglyphs lined the walls—probably was the ASCII of the time.

As with all BSD conferences, I create a schedule in advance of the talks I want to attend. But once I'm there, I sometimes get into an interesting discussion with people and miss the start of a talk. At other times, I make up my mind on the spot and wander into the exact opposite talk—with mixed results. Recordings permit me to watch the talks I missed once they are available, but nothing beats hearing a good talk in person.

On the first day, I missed the battle of the BSDs talk by Antoine and Baptiste because I had to proctor the BSDA exam. I saw the first part at AsiaBSDcon this year, so I did not miss it completely, and giving people a chance to get their BSD knowledge certified is as good an excuse as any for missing it.

The social event was held on a boat next to the Eiffel Tower. Instead of a traditional dinner, where you sit down at either a single table with a few people or a very long one, this was a standing dinner. Waiters walked around all night serving delicious snacks that only the French can produce. There was

also an open bar. Groups formed and split up again, allowing for varied conversations with a lot of people. Also, since the boat was moving and night had fallen, we saw a panorama of illuminated Paris at night. At times, we passed under a bridge that was just an arm's length above our heads. Sometimes other boats overtook us, or we would see people dancing along the riverfront. It was a wonderful night, with many good conversations about BSD and other topics.

The Conference Day 2

The next conference day was just as good as the first. I helped out at the FreeBSD Foundation table, talking to people and making new acquaintances. Many of them knew my name before I knew theirs, as they watch BSDNow, where I have been a host since Kris Moore's retirement from the mic, providing news about what's going on in the BSDs. I enjoyed the feedback and encouragement.

Before I knew it, the closing keynote by Brendan Gregg was starting. I was seated in the front row because I had a job to do during the closing session. Brendan had revised some of his slides about performance analysis methodologies, but also had new slides centered around BSD tools. That way, it provided something new for everyone. The talk is available here: http://www.brendangregg.com/blog/2017-10-28/bsd-performance-analysis-methodologies.html.

During the closing session, various future conferences were pitched, and the audience was encouraged to attend and submit talks. The various BSDs gave short presentations about their work, and Deb Goodkin and I recognized the work of four FreeBSD developers.

Both Baptiste and Antoine, who had worked so hard to put together this event, were rather hoarse by the end, but everyone thanked them for their hard work and dedication in making this EuroBSDcon such a great success. I went with a small group to have one last dinner in Paris and to reflect on the conference. Some people stayed in France a couple of days, but for me, I had to fly home, knowing that this had not been my last EuroBSDcon. ●

BENEDICT REUSCHLING joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He is a proctor for the BSD Certification Group and joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany.

2018 Events Calendar

The following BSD-related conferences will take place during the first quarter of 2018.

FOSDEM • Feb. 3 & 4 • Brussels, Belgium

https://fosdem.org/2018/ • This annual two-day event promotes the widespread use of free and open-source software. There will once again be an all-day BSD Devroom on February 3. This event is free to attend.

AsiaBSDCon • March 8–11 • Tokyo, Japan

https://2018.asiabsdcon.org/ • This is the annual BSD technical conference for users and developers on BSD-based systems. It provides several days of workshops, presentations, and a Developer Summit. Registration is required.

SCALE • March 8–11 • Pasadena, California

http://www.socallinuxexpo.org/scale/16x • The 16th annual Southern California Linux Expo will once again provide several FreeBSD-related presentations and a FreeBSD booth in the expo area. This event requires registration at a nominal fee.